Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT 1) Approved for public release; distribution is unlimited. ~~2) (continued on reverse side)~~
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Electrical Eng. Georgia Tech 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION U.S. Army Strategic Defense Command
6c. ADDRESS (City, State, and ZIP Code) Atlanta, Georgia 30332	7b. ADDRESS (City, State, and ZIP Code) P.O. Box 1500 Huntsville, AL 35807-3801
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DASG60-85-C-0041
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Macrostructure Logic Arrays Volume 1

12. PERSONAL AUTHOR(S)
C. O. Alford

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 6/28/85 TO 11/2/90	November 2, 1990	390

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Vol. 1 Task 1: Digital Emulation Technology Laboratory

1. Introduction
2. Hardware and Facilities
   - 2.1 Parallel Function Processor
   - 2.2 Seeker Scene Emulator
   - 2.3 Other computer systems
   - 2.4 Secure Laboratory
3. System Software
   - 3.1 Floating Point Processor
   - 3.2 Crossbar Sequencer Compiler
4. Application Software
   - 4.1 Spinning Missile
   - 4.2 KWEST/EXOSIM
5. Appendices

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

FINAL REPORT

VOLUME 1

TASK 1: DIGITAL EMULATION TECHNOLOGY LABORATORY

CLIN 0006

November 2, 1990

MACROSTRUCTURE LOGIC ARRAYS

Contract No. DASG60-85-C-0041

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

Contract Data Requirements List Item F006

Period Covered: 1985-1990

Type Report: Final

DISCLAIMER

DISCLAIMER STATEMENT - The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

DISTRIBUTION CONTROL

(1) DISTRIBUTION STATEMENT - Approved for public release; distribution is unlimited.

(2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227 - 7013, October 1988.

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT 1) Approved for public release; distribution is unlimited. 2) (continued on reverse side)
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Electrical Eng. Georgia Tech 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION U.S. Army Strategic Defense Command
6c. ADDRESS (City, State, and ZIP Code) Atlanta, Georgia 30332	7b. ADDRESS (City, State, and ZIP Code) P.O. Box 1500 Huntsville, AL 35807-3801
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DASG60-85-C-0041
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)

Macrostructure Logic Arrays Volume 1

12. PERSONAL AUTHOR(S)
C. O. Alford

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Final	FROM 6/28/85 TO 11/2/90	November 2, 1990	390

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Vol. 1 Task 1: Digital Emulation Technology Laboratory

1. Introduction
2. Hardware and Facilities
   - 2.1 Parallel Function Processor
   - 2.2 Seeker Scene Emulator
   - 2.3 Other computer systems
   - 2.4 Secure Laboratory
3. System Software
   - 3.1 Floating Point Processor
   - 3.2 Crossbar Sequencer Compiler
4. Application Software
   - 4.1 Spinning Missile
   - 4.2 KWEST/EXOSIM
5. Appendices

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

Distribution statement continued
2) This material may be reproduced by or for the U.S. Government pursuant to the copyright license under the clause at DFARS 252.227-7013, October 1988.

FINAL REPORT

VOLUME 1

TASK 1: DIGITAL EMULATION TECHNOLOGY LABORATORY

CLIN 0006

November 7, 1990

Authors

Thomas R. Collins, Stephen R. Wachtel

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30332 - 0540

Eugene L. Sanders
USASDC
Contract Monitor

Cecil O. Alford
Georgia Tech
Project Director

Table of Contents

1. Introduction

The Digital Emulation Technology Laboratory (formerly referred to as the KEW Digital Emulation Laboratory) is a principal unit within the Computer Engineering Research Laboratory (CERL) at Georgia Tech. This report addresses the objectives, requirements, and schedule of the Digital Emulation Technology Laboratory (DETL), relative to contract number DASG60-85-C-0041. An associated report, "Annual Report -- Task 1: Digital Emulation Technology Laboratory" covers DETL relative to contract number DASG60-89-C-0142. The major distinction between these two contracts and their associated activity at DETL is that the newer contract concerns primarily activity associated with the effort to develop an integrated hardware and software environment for end-to-end simulations of exoatmospheric interceptors such as EXOSIM. This report, on the other hand, focuses more on the basic hardware and system software that was developed at Georgia Tech and eventually applied to these end-to-end simulations. This includes the Georgia Tech Parallel Function Processor (PFP), the older system software for the PFP (utilities and parallel programming tools), and earlier application software (prior to EXOSIM).

1.1. Objectives

Within DETL, there are two main hardware systems: the Parallel Function Processor (PFP) and the Seeker Scene Emulator (SSE). Each of these systems is a complex parallel processor, designed to function together as an emulation facility for kinetic energy weapons systems. Software development is also an active area of research, both at the system level (compilers, loaders, graphics development) and at the application level (simulation and emulation studies).

The principal objectives of DETL are as follows:

- Provide facilities for 6-DOF KEW emulation

- Provide real-time capability in excess of 2000 Hz

- Provide real-time emulation of IR FPA seekers
- Test and verify GN&C software and hardware systems
- Educate new PFP users and provide technical support.

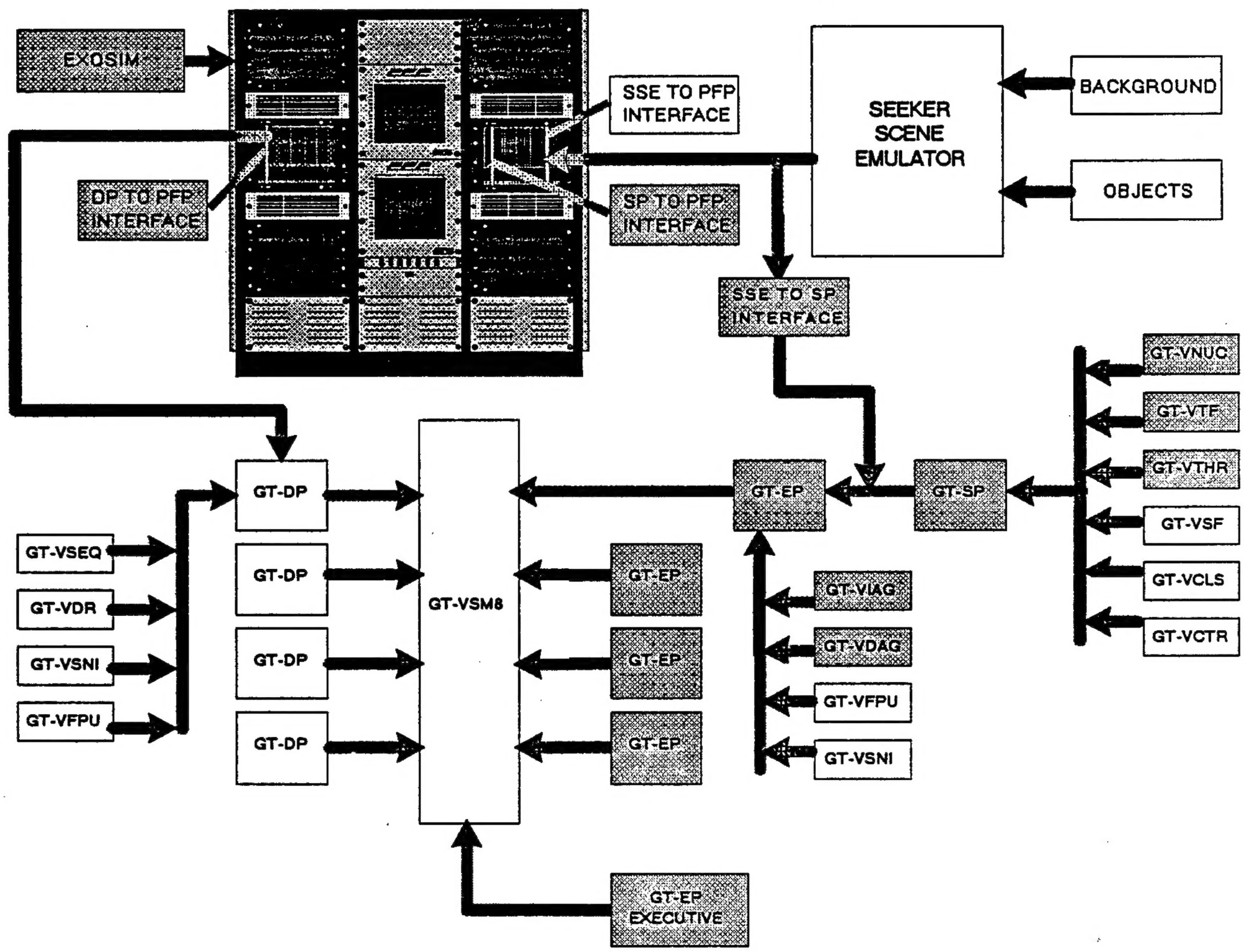

Figure 1.1: Major components of DETL

The major components used in meeting these objectives include the PFP, SSE, high-speed 3-D graphics workstation, and associated conventional computers for basic support functions. Not all of these components are required for every task. For example, much of our work consists of running simulations (sometimes real-time, sometimes not) on the PFP, with no attached systems. This limited mode of operation is capable of verifying missile simulation models and control laws, as well as many types of signal processing.

To provide realistic imagery in real-time, however, the Seeker Scene Emulator is required. This system generates image data as though it were coming directly off of the elements of a focal-plane array, with the scene information determined by the relative location of the simulated missile system to the targets and decoys.

Actual flight hardware may be tested within this system, as indicated by Figure 1.1. Most of the items contained in the lower half of this figure represent VLSI components that may be tested within DETL. The GT-DP blocks, for example, are chips for guidance and control processing that are being developed at Georgia Tech. Similarly, the GT-SP block contains signal-processing components developed at Georgia Tech. By equipping the hardware with appropriate interfaces to the PFP, the simulated functions of the GN&C Processor can migrate from the PFP to the actual hardware. These interfaces are also shown in the figure. Additional detail on the VLSI components themselves may be found in Volume 4 of this final report..

1.2. Schedules and milestones

As of July 1990, there are three 32-processor PFP systems available. One of these is currently undergoing a transition from an earlier configuration to our latest 3-processor-rack configuration, with 386-based processors to replace the original 286-based processors. The other two systems are the 286-based machine allocated for KDEC and the FPP-based machine for internal development of FPP/Sun host software. Not included in these 3 PFPs are a limited test PFP system and the prototype Multibus II PFP system. Since last year, we have taken our older 8086-based PFP out of service.

The FPP-based PFP and the KDEC PFP both include the basic packaging and power supplies to support expansion to 64-processor capability. The 386-based PFP may eventually be paired with the Multibus II PFP to produce a 64-processor hybrid system.

The major milestones completed over the period of this report are as follows:

- Integration of 386/12 processors into the PFP, making the 286/12 processors available for the KDEC PFP,

- Demonstration of the 256-processor Seeker Scene Emulator generating frames at 64 frames/second, with image control information supplied by the PFP in real time,

- Upgrades to the crossbar compiler to support multiple crossbars,

- Transition to a new host operating system (RMX II), allowing greater memory accessibility, virtual terminal support, and other features,

- Upgrades to the Floating-Point Processor (FPP) Compiler to support migration to a new host,

- Development of utility software on the new system, replacing (and enhancing) basic functions for loading and starting programs,

- Development of programming tools for the new system, including a "make" utility for application maintenance,

- Development of parallel-processing support utilities, including one that analyzes variable usage across partitions and one that automatically generates communication calls,

- Development of libraries of communication procedures for processor-processor and processor-host interaction, providing uniform interfaces across several languages (C, Fortran, Pascal, and PL/M),

- Design and development of a new "piggyback" board to provide crossbar communcation capability to the 386/12 boards through their iSBX interfaces,

- Design modifications to the Multibus Repeater boards to support expanded memory accessibility (16 MBytes per rack, over 48 MBytes per 32-processor system),

- Design modifications to the 286/12 processors to make them completely interchangeable with 386/12 boards in the new, expanded-memory configuration,

- System-level repackaging (racks, power supplies, cabling),

- Presentation of onsite education in PFP programming,

- Hardware and software documentation for the PFP Technical Data Package,

- Definition of basic rules for developing parallel applications in FORTRAN and in ACSL on conventional single-processor systems,

- Application of these rules in versions of EXOSIM by Dynetics, followed by successful porting of 4- and 5-processor versions to the PFP.

- Development of new firmware for 286/12 boards to support their use as target processors,

- Pascal compiler running on FPPs.

Some of these items are more closely related to the work effort of the new contract (DASG60-89-C-0142), and are therefore described in that report.

1.3. Long-range Plans

Georgia Tech's parallel architectures continue to evolve. Already in progress is the development of a PFP based on a newer computer bus, the Multibus II. With this high-performance bus, an additional interconnection path is available between processors, more suitable for occasional transfers of large amounts of data. A new interconnection scheme between racks of processors will allow multiple PFPs to be interconnected more freely than ever before.

Beyond the PFP, a new dynamic crossbar architecture is in the design stage. Unlike the current crossbar, which pre-schedules all of the communication between processors, the new architecture will allow processors to communicate at their own pace, even changing the system to meet each processor's changing requirements.

Such an architecture is suitable for more general problems, like the simulation of molecular dynamics or compressible fluid flow. It will also be the best architecture to handle SDI's battle management problem. In this application, a computer must respond quickly to a variety of potential threats. A dynamic crossbar system would provide both the required number of processors and the fast communication paths.

Although we have already developed a complete set of software tools that meet our own research requirements, there is a need for general programming aids, particularly for users not accustomed to specialized computers. A graphic interface has been prototyped which provides users with the

capability of entering simulation models directly as block diagrams. Other tools would automatically partition large problems among the available processors.

At the same time, we will continue to apply the best available chip technology to our designs, including both VLSI and VHSIC, where applicable.

1.4. Report Summary

The remainder of this report will describe the hardware and software associated with the Digital Emulation Technology Laboratory, with an emphasis on the work completed during the previous contract year. The hardware information includes updated status of the PFP units, new processors, host enhancements, communication interfaces, improved coprocessor performance, and new firmware. A brief description is also given for the physical facility itself and some auxiliary computers contained within. The software information includes new versions of utilities which support the GT-FPP (Floating-Point Processor) and the GT-XSD and GT-SEQ (crossbar and sequencer boards), as well as updates to application software (the Spinning Missile simulation and EXOSIM).

2. Hardware and Facilities

This section begins with a description of the Parallel Function Processor, including recent changes, and then discusses the current configuration of the two alternative host computers. Most of the detail, though, is devoted to recent improvements, including the upgrade from 286-based processors to 386-based processors. This not only required configuration and testing of the new 386-based boards (Intel iSBC 386/12 boards), but also modifications to the attached crossbar interface board, the onboard firmware, and the Multibus repeater subsystem. At the same time, coprocessor performance was enhanced by the selection of the Cyrix device, rather than the standard Intel 80387, but some compatibility problems were found. These issues, along with current board status, are also covered.

2.1. Parallel Function Processor (PFP)

The Special-Purpose Operational Computing Kernel, or SPOCK, evolved from a Ph.D. dissertation (by James O. Hamblen) on a new architecture designed to solve ballistic missile simulations. Before digital computers came into prominence, some of these simulations had been performed quite effectively on analog computers, in which basic circuit elements are interconnected by a patch panel to create an approximation to the real system.

Digital computers provided the potential of much higher accuracy in the simulations, but at the cost of speed: most real systems could not be simulated nearly as fast as they really run, generally referred to as *real time*. In 1978, Georgia Tech's SPOCK I addressed the problem by showing how up to 6 processors could effectively perform such a simulation.

Building on the previous experience, in 1982, a prototype of a 32-processor system, SPOCK II, demonstrated greater capability with more-powerful processors. In addition to the digital processors, SPOCK II also had analog input and output channels. This provided the important capability of interfacing seamlessly with the external environment, for real-time control of analog systems.

Since that time we have developed SPOCK II into the Parallel Function Processor (PFP), a fully-operational testbed for simulation problems from both military and nonmilitary applications. The architecture never stagnates -- the original Intel 8086/8087 processors were each roughly as powerful as an IBM PC, but now they can be replaced with any of three newer processors. One is based on the Intel 80286/80287 and performs as well as an IBM AT. Another is based on the Intel 80386/80387, and the last is based on the AMD 29325 and is about 25-100 times faster than the other processors for the floating-point calculations which it is designed to perform.

All of the processors support the 16-by-16 crossbar interconnection, allowing each to communicate directly with the others. Multiple conversations may take place simultaneously on the crossbar, and it is also possible for a single processor to broadcast data to every other processor in a single instruction cycle. Since the crossbar has been reduced in size from a full 19-inch rack down to a cluster of eight circuit boards, it is now possible to have the power of 32 minicomputers in two racks, and still have all of the processors work together efficiently.

Each of the current processors has two interfaces: one to the crossbar for data communication while running, and one to a shared bus that is used for loading programs and data from a central host. Virtually any imaginable processor can be fitted to a processor slot in the PFP. This includes multiprocessors, like the array processors described earlier. So, if an image processing problem was part of a larger simulation problem, it could be assigned to an array processor within the PFP system. Co-processing boards have been developed at Georgia Tech that evaluate complex floating-point functions in a fraction of the time used by the best supercomputers on the market today. These co-processing boards "piggy-back" on the processors described earlier.

Similarly, a complete minicomputer system with an attached 3-D graphics workstation has been connected to one of the PFP processors, thus effectively becoming a part of the multiprocessor system. This allows sophisticated graphics to be generated in real time as the simulations proceed.

These enhancements demonstrate that other architectures can be applied *as needed* within the enveloping PFP architecture. But there is also a way to increase the PFP's capability at a higher level. Since the number of processing nodes in a crossbar is practically limited because of the large number of switches required, the PFP needs a way to grow beyond its crossbar. A fully-operational interconnection board has been developed which occupies a processor node in a single PFP system. When a processor communicates with this interconnection board, the data is

passed out over an external channel to an identical board in another complete PFP. By adding more interconnections, multiple PFPs may form a higher level of parallel processing. A triangle of three PFPs still allows each processor to communicate with any other processor with no intervening processors, although there may be some waiting for an available channel.

The standard configuration of the PFP at this time is a 64-processor system (2 crossbars), packaged in a three-rack system, including the host. A single-crossbar system can be packaged in two smaller racks, if desired. A 64-processor system fitted with custom interfaces is planned to be installed at Arnold Engineering Development Center in Tennessee to support simulation studies.

Currently, a new host computer for the PFP is being developed, based on the Sun 386i. This new host will provide enhanced support for graphic input and output, and it is now serving as a platform for the development of an ADA compiler for the PFP.

2.1.1. Physical Description

The full 64-node PFP, complete with the host computer, occupies three 19 inch wide by 32 inch deep by 75 inch high equipment racks. Each outer rack contains 32 PPE slots. The center rack contains the two crossbars, two sequencers, the host computer and two crossbar status displays.

All processors, as well as the sequencer, conform to Intel's Multibus I specification. They are connected to the host through a custom Multibus repeater system, which is used by the host to communicate with each PPE. Each 16 by 16 crossbar switch is made from four 8 by 8 switch boards connected through a custom backplane. Each 8 by 8 switch board is built to a 15.75 inch by 14.44 inch Eurocard standard. Both crossbars are housed in one 19 inch wide card cage.

Each of the 64 nodes in the system is occupied by a PPE. A PPE can be one of five different boards; an array interconnect, an Intel 80286-based commercially available processor, an Intel 80386-based commercially available processor, a Georgia Tech Floating-Point Processor, or a multi-channel analog I/O interface. Other boards will be developed as necessary to enhance the capability of the PFP.

The Georgia Tech Floating Point Processor (GT-FPP/3) is an 8 MFLOP computing engine based on the AMD 29325 floating point chip. Currently, the board is programmed using a subset of Pascal. An ADA compiler is in development for use with this processor.

The iSBC286/12 processor is commercially built by the Intel corporation. It is a cheaper, lower performance board than the GT-FPP/3. The board is useful in applications that require large amounts of memory such as table look ups. Presently, most of the programming is done in Pascal or PL/M, although FORTRAN, C, and other Intel standard utilities are available. The crossbar interface to this board is built to fit the Intel standard iSBX port. Supporting other Multibus I processors that have this port only require changes in the board's firmware. The iSBC386/12 processor is an 80386-based equivalent of the iSBC286/12 board, with approximately a 3-4 times speed improvement for typical PFP applications.

The analog input/output board consists of four analog to digital input channels and four digital to analog output channels. The output portion consists of 4 separate digital to analog converters. The input portion consists of 4 sample and hold circuits multiplexed through one analog to digital converter. Any combination of inputs and outputs are available for use. All digital conversions have 12 significant bits.

As previously mentioned, the array interconnect (GT-ARI/1) is used as a direct interconnect between crossbars. Each array interconnect may send and receive 16 bit words simultaneously from other array interconnects.

All programs are written and compiled on the host computer then down loaded to the processors. Currently, each problem is analyzed by a programmer and split into parts which are then compiled for individual processors. A separate compiler is used to load the crossbar and sequencer with the instructions for processor communication.

The major components of a full system are:

1. The host machine. (This may be an Intel 310 or Sun 386i)

2. An MDB Systems Data Shuttle 2000 removable disk drive unit.

3. Up to 64 processors and array interconnects, in any combination.

4. Up to two sequencers.

5. Up to two full 16 by 16 GT-XB/2 crossbar switches.

6. Up to two GT-XSD/2 status display units.

7. Up to two equipment racks containing Multibus I card cages, sequencer cabling, and power distribution.

8. One equipment rack containing the crossbar, sequencers, crossbar status displays, and appropriate power distribution.

2.1.2. Intel 310 Host

The Intel 310 host is based on a 12 Mhz 80286 processor (actually the same 286/12 board available for use in the PFP) and runs the Intel iRMX operating system. We have recently verified that it is possible to replace the host 286/12 board with a 386/12 board, in much the same way that we have replaced the PFP 286/12 processors with 386/12 processors. This configuration can execute computationally-intensive applications (like compilation and linking) about four times faster that the 286-based host. The host is tied to the PFP through a custom set of repeater boards developed here at Georgia Tech. A master repeater board is located within the host chassis, and slave repeater boards are located within the racks of processors. The machine supports all standard Intel languages running under the iRMX operating system, including C, Pascal, PLM, and FORTRAN. Programs written in any of these languages may be compiled and linked on the host and then downloaded to processor boards (iSBC 286/12s or iSBC 386/12s) in the PFP for execution. In addition, the host supports a compiler that implements a subset of Pascal for use with the GT-FPP/3 custom floating-point processor.

2.1.3. Sun 386i Host

The Sun 386i host is based on a 25 Mhz 80386 processor and runs the Unix operating system. It is the eventual replacement for the Intel 310, leading to higher performance and a more user-friendly environment. The hardware interface to the PFP is similar to that of the Intel 310 host, except that the master repeater board is located within a dedicated Multibus rack, connected to the Sun host by a PC-to-Multibus link. (The Sun 386i utilizes the PC/AT bus.) A C compiler is being written to support the GT-FPP/3 processor, and other languages will be supported via translators (Ada-to-C and FORTRAN-to-C). All low-level drivers interfacing the Sun to the PFP are complete and several small Fortran, Ada, and C programs have been loaded and tested.

Eventually, the Sun will also support standard languages for programming the iSBC 386/12 processors.

2.1.4. iSBC 386/12 Processor Integration

As noted earlier, the iSBC 386/12 processors are faster replacements for the iSBC 286/12 processors. The integration of the 286/12 processors was completed during a prior year of this contract, and no additional developments have occurred in this area. The 386/12 processors, however, were integrated mostly over the period of early 1990. This involved several tasks, including:

- determination of proper board configuration,
- Multibus repeater modifications to support expanded address space
- development of monitor PROM firmware,
- development of loader software,
- testing of iSBX crossbar interface,
- re-layout of iSBX crossbar interface,
- testing, debug, and workarounds for Cyrix coprocessor, and
- configuration and testing of full complement of 32 processors.

It was during the integration of these processors into the system that we decided to upgrade our host operating system from RMX I to RMX II. This allowed us to access a total of 16 Megabytes of memory within each rack of processors (at least 1 Megabyte per processor). It also allowed us to use 80286-based and 80386-based compilers and utilities for future application development on the PFP. Most of this development that occurred under the RMX II host is covered in the other final report for the Digital Emulation Technology Laboratory, which emphasizes the newer tools and applications. Because of the transitional nature of the hardware and firmware modifications, however, they are covered within this report.

2.1.4.1. iSBX Crossbar Interface Boards

One major factor in the 286-to-386 transition was the usage of the iSBX Crossbar Interface Board (GT-XI286/2), which had some effect on several of the items listed above. The earliest PFP processors, the Intel 86/12 boards, utilized a specialized interface to the crossbar that plugged into a ROM socket. In the interests of standardizing on an interface which would be adaptable to the 286/12 boards and follow-ons, a new crossbar interface board was developed, based on the standard iSBX interface. This standard interface was established by Intel and is available for "piggy-back" functionality on a wide range of Multibus I and Multibus II processor boards.

The iSBX I/O Expansion Bus specification is equivalent to the proposed IEEE standard P959. Detailed descriptions may be found in Intel product data books and in the Intel publication *Intel iSBX Bus Specification* (manual order number 142686-001).

Briefly, the iSBX bus supports both 8-bit and 16-bit data transfers at a maximum rate of 1 transfer per microsecond. The controlling processor board generates 3 low-order address bits and 2 select signals (high byte, low byte, or both), allowing the attached iSBX board to contain up to 16 addressable bytes. Typically, these blocks are defined as registers which are offset from some base I/O address in the processor's address space. On the 286/12, the GT-XI286/2 board generally resides at base address 80H (in the I/O space, not the memory space), with the data register at 80H/81H and the status register at 82H/83H.

The GT-XI286/2 board does not support interrupts or DMA. It *does* use the clock signal and reset signal, and it does *not* extend the transfer cycle through the use of MWAIT/. The GT-XI286/2 board was designed according to the physical and electrical specifications for a double-width iSBX board. (The specification allows for two sizes of boards, and we required the double-width form-factor, mainly to accomodate ribbon-cable connectors.) Frequently, processor boards provide two iSBX connectors, allowing the user to attach up to two piggy-back boards. This was the case with the 286/12 board, which was desirable for the PFP application, since it allowed for both a single-width iSBX board (like the Georgia Tech Function Board, which evaluates complex arbitrary functions of a single real number), as well as the double-width GT-XI286/2 board.

Although the 386/12 board was supposed to be a drop-in replacement (after proper configuration) for the 286/12 board, we soon encountered a major discrepancy. The 386/12 board had two iSBX connectors, but it was only possible to use two standard iSBX boards if both were single-width. Since the GT-XI286/2 board could not be reduced to single width, it was redesigned with a non-standard form-factor that avoids the memory module.

We were able to test the 386/12s with the original iSBX design by plugging the board onto the only connector which accomodated a double-width board. This merely prevented us from using the Function Board or any other iSBX board during the test period. Once the operation was verified, the physical layout of the board was altered by Intel at no charge, as partial compensation for what we perceived as a design flaw. The new version of the iSBX Crossbar Interface Board is part number GT-XI286/3.

During extended testing, we discovered intermittent communication errors with this board. The problem was eventually traced to electrical noise due to inadequate power and ground distribution. The board had been designed as a double-sided board, with no internal planes for power and ground. Although the power and ground traces were wider than signal traces, they were apparently not sufficient. We were able to repair the boards by adding two wires to improve distribution along the long axis of the board. All boards have continued to perform satisfactorily in extended testing. Before we order any more of these boards, however, we will re-layout the board with internal power and ground planes.

2.1.4.2. Board Configuration

The board configuration consists of installing EPROMs (see Monitor Firmware section), installing a coprocessor (see Cyrix Coprocessor section), installing the iSBX Crossbar Interface board, and setting jumpers. The 386/12 boards purchased for the PFP were configured with one Megabyte of memory, although it is possible to use more memory on at least some of the processors. The default jumper settings for these one-Megabyte boards are suitable as a starting point for configuration. These settings are listed in the "iSBC 386/12 Hardware Reference Manual," available from Intel.

The deviations from these default jumper settings are as follows:

- iSBX DMA is disabled (DMA is not required),

- PROM size is 256K,

- Dual-port memory is set to allow the full Megabyte to be accessible at one of 11 locations, depending on where board is to be installed,

- Two onboard timers are cascaded,

- Onboard processor is allowed RAM access all the way up to ROM start (no outward Multibus window),

- Coprocessor is enabled,

- Board ID code changed to reflect coprocessor presence, and

- iLBX local memory bus extension is disabled.

2.1.4.3. Monitor Firmware

One of the principle reasons that we made the switch to the RMX II operating system and the 386/12 processor boards was so that we would have the full benefit of a larger working memory space on each target, running under the *protected* mode of the 386 processors. (This would have also been an advantage with the 286/12 processors which we had been using for some time.) The monitor which we had been using on both the 286/12 and 386/12 processors supported only the *real* mode of operation, which is limited to an emulation of the simpler 8086 family of processors, with little capability to manage and control memory segments. It was therefore necessary to develop a new monitor.

This new monitor was based on Intel's iSDM monitor, which is available as Intel product number SDMSC, version 3.2. This product allows a user to begin with a basic monitor which can be configured for a custom application by using various macro calls. It also allows customized user code to be inserted within the monitor, which was necessary for our application.

Some of the advantages of the new monitor (and protected-mode operation) include:

- greater flexibility in choice of program start address,

- ability to create specialized buffers for host-processor communication, and

- automatic creation of read-only code segments and multiple data segments

The code for the user-defined section of the monitor firmware is included in Appendix A.

2.1.4.4. Multibus Repeater Modifications

The 386/12 processors, like the 86/12s and 286/12s which preceded them, provide dual-port RAM (memory which is accessible by both the onboard processor and any bus master on the Multibus). In the PFP, this access from the bus is needed by the host processor and its associated Multibus repeater system. The Multibus repeater system essentially allows multiple Multibus card cages to map into the address space of the Multibus-based host. In order to fully utilize the one Megabyte of dual-port RAM on the 386/12 processors, it was necessary to modify the repeater system. Previously, it was only possible to have a maximum of one Megabyte of memory in each PFP processor rack, and this had to be split up among the processors in that rack. Typically, this resulted in only 64K or 128K of accessible memory on each processor, which at this time was a 286/12.

As work on advanced simulations such as EXOSIM proceeded, this memory limitation became critical. Anytime the code exceeded 64K, a special loader function would be required to access the additional 64K on those boards that had a total of 128K available. It also appeared that some of the partitioned code would even exceed the 128K limit. Furthermore, there was no straightforward means of configuring the memory on the newer 386/12 boards to map only 64K or 128K in a distinct location of the Multibus address space. (This was another unforeseen incompatibility between the 286/12 and 386/12 boards.) The solution to this problem was to increase the dual-port RAM to a full Megabyte.

Modifications were made to PALs on the 286 processor boards and on the slave repeater boards, and some wire cuts and adds were made on the slave repeater boards as well. We also installed the Multibus "P2" connector on the card cages in this system. This connector contained only signals which were never required in previous configurations (at one point, cables were routed out of the backplane at the P2 location). The uppermost bus address lines, needed to access the full 16-Megabyte address space, were included on this connector, so we had to install it at this point.

2.1.4.5. Cyrix Coprocessor

We ran into some unexpected problems when we attempted to run some applications on multiple target processors in the PFP. Surprisingly, we were unable to run it successfully even on a single target processor. The problem was eventually traced to the Cyrix math coprocessor chips on the 386/12 boards. These are supposed to be direct replacements for the Intel 80387 chips, but unexplained errors kept coming up. Once we switched back to the Intel chips, the program ran normally. We wanted to use the Cyrix chips, since we could get a performance improvement of up to 20% over the 80387.

We received the most current revision of the Cyrix chips and tried to use them, but we found no difference. We also tried the Cyrix chips in a 386 PC-compatible machine, also running EXOSIM. In this configuration, we had no errors. Cyrix also sent a socket adapter with a built-in PAL that fixes a problem in certain configurations (probably in certain PC-compatibles). We tested this fix, too, and it made no difference in our application.

We continued to explore the apparent incompatibilities between the Cyrix coprocessor and the 80387 that it is designed to replace (with a performance improvement of up to 50%). We were eventually able to trace the problem to a misalignment of the internal stack in the coprocessor. The Intel FORTRAN compiler generates code which uses reserved (and supposedly unimplemented) 80287/387 instructions. These instructions, however, are actually implemented in the 80287 and 80387 hardware as redundant ways to accomplish the same effect as normal, documented instructions. Cyrix, however, did not emulate these reserved instructions, since it should not have been necessary to do so. (The Intel compiler generated these reserved codes, when it should have generated equivalent functions using docutmented instructions.) The workaround to the problem is to always turn off optimization in the FORTRAN compiler. This prevents the unimplemented instructions from ever being generated.

The five-processor version of EXOSIM (described briefly later in this report) was recompiled using the new compiler, then run on Cyrix-equipped processors. The timing results are given in Table 1. For real-time performance, the execution time for each partition must be 1.0 sec/real-time sec or less. This particular partitioning was an early attempt and does not represent our best effort for EXOSIM. For comparison, the same partitioning was run on the Intel 80387 coprocessors, and the results are given in Table 2.

Table 1: EXOSIM five-processor version (new compiler, Cyrix chips)

Processor	1	2	3	4	5
Sec /real-time sec	8.588863	1.912137	1.698079	1.032199	0.596258

Table 2: EXOSIM five-processor version (previous compiler, Intel 80387s)

Processor	1	2	3	4	5
Sec /real-time sec	11.1534	3.54096	2.47603	1.47417	0.873123

As these results indicate, performance had been improved by up to 46% (but only 23% on the "bottleneck" partition). We continued to use this workaround to avoid further problems with the Cyrix coprocessor, and we have been satisfied with the improved performance.

2.1.4.6. Board Testing and Status

After we had determined a satisfactory configuration for the 386/12, including monitor, coprocessor, dual-port configuration, and iSBX usage, we tested a limited number of boards in a test PFP that we had developed specifically for that purpose. During this time, we had only five to seven 386/12 boards available, but this was sufficient to develop test software, as well as to experiment with some simulation applications. By the time we were satisfied with the performance and reliability of the 386/12s, we had received a full complement of 32 boards.

Each of these boards was configured and tested, then subsequently burned in for millions of crossbar transfers. Some boards failed initial testing, and others failed during the burn-in period. These were returned to Intel for repair, mostly during the warranty period. The current status of these boards is given in Table 3. The "location" column gives either a general location, such as the Intel repair facility or the Sun development PFP, or a specific location in the 386-based PFP. These specific locations include a letter and a hexadecimal number. The letter is either T, M, or

B, for Top, Middle, or Bottom rack. The hexadecimal digit is the base address page of the current dual-port memory setting, where "8" would indicate a base address of 800000 (hexadecimal).

Table 3: 386/12 Board Status

Serial Number	Order*	Condition**	Coprocessor	Location
N00209451	1	OK	Cyrix	B2
P00448691	2	OK	Cyrix	T6
P00448744	1	OK	Cyrix	Sun
P00466255	3	OK	Cyrix	T1
P00466256	3	OK	Cyrix	Sun
P00466258	3	OK	Cyrix	Sun
P00466261	3	OK	Cyrix	M5
P00466262	2	OK	Cyrix	T4
P00466264	3	Failed in Sun		Intel
P00466357	3	OK	Cyrix	T2
P00466358	3	OK	Cyrix	M9
P00547569	2	Failed in PFP		Intel
P00568416	3	Failed in Sun		Intel
P00568420	3	OK	Cyrix	M6
P00568471	3	OK	Cyrix	T7
P00592160	3	OK	Cyrix	M8
P00593319	2	OK	Cyrix	B8
P00610398	3	OK	Cyrix	T9
P00610405	2	OK	Cyrix	T3
P00610406	2	OK	Cyrix	T5
P00610479	2	OK	Cyrix	Sun
P00638698	3	OK	Cyrix	M3
P00638699	3	OK	Cyrix	MA
P00638700	3	OK	Cyrix	M7
P00638702	3	OK	Cyrix	TB
P00638703	3	OK	Cyrix	M2
P00638704	3	DOA twice		Intel
P00638705	3	OK	Cyrix	B1
P00638710	3	OK	Cyrix	TA
P00638712	3	OK	Cyrix	B7
P00638713	3	OK	Cyrix	M4
P00638724	3	OK	Cyrix	MB
P00641883	3	OK	Cyrix	T8
P00641887	3	OK	Cyrix	M1

* 1- First two boards ordered for testing
2 - First seven boards received as part of large 32-board order
3 - Remainder of large order

** OK - Memory, crossbar port, coprocessor all checked out
Failed - Initially OK, but subsequently failed
DOA - Failed initial testing

2.2. Seeker Scene Emulator (SSE)

In addition to developing crossbar machines like the PFP, we are actively studying other architectures, since there is no such thing as a completely general-purpose parallel computer. One of the most promising is a group of architectures built around a new microprocessor chip, the Inmos Transputer. Unlike previous microprocessors, the Transputer was specifically designed to be interconnected with others of its kind. Since a single chip includes the processor, memory, and communication ports, it is possible to build a parallel machine with little more than a group of Transputers.

Each Transputer has four links that can be used to tie them together, allowing a wide range of architectures to be built. One of our principal applications for the Transputer is a Seeker Scene Emulator, a machine that models what an imaging sensor on a missile would see during a mission. Most simulations of such systems tend to simplify the infrared sensing process in order to minimize computations, but the Georgia Tech Seeker Scene Emulator will provide a signal which can be displayed on a screen and will look virtually identical to a real view of an incoming threat.

This seeker output can then be used by a simulation running on the PFP, or by an actual guidance and control processor, like the one being developed for our VLSI devices. The Seeker Scene Emulator will use 256 Transputers, so when connected to PFP in a simulation, it will be another example of a specialized parallel processor within the more general crossbar architecture of PFP.

Under direction from the U. S. Army Strategic Defense Command, the Computer Engineering and Research Laboratory at the Georgia Institute of Technology and BDM Corporation are developing a real-time Focal Plane Array Seeker Scene Emulator. This unit will enhance Georgia Tech's capabilities in KEW system testing and performance demonstration.

The FPA Seeker Scene Emulator combines advanced hardware developed at Georgia Tech with a BDM-generated database to produce signals based upon target radiometric information, seeker optical characterization, FPA detector characterization, and simulated background environments. Using real-time, positional updates, typically from the Georgia Tech Parallel Function Processor, the Seeker Scene Emulator can combine elements of the pre-computed database to form an image that is positionally and radiometrically correct.

In conjunction with development of the FPA Seeker Scene Emulator, research into signal processing of seeker data is underway. The Seeker Scene Emulator provides a platform for the expedient testing of algorithms and implementations. Currently, a parallel-processing network is being used to test various signal processing "building blocks."

Detailed information about the Seeker Scene Emulator may be found in a separate final report.

2.3. Other computer systems

Currently a Digital Equipment Corporation MicroVax II is used as the primary file server for the Seeker Scene Emulator. This system is equipped with a nine-track tape system, a high-density EXABYTE tape cartridge unit, the standard TK50 tape unit, an Ethernet network interface, and a Caplin Cybernetics Corporation QT0 Transputer Interface Module. Using the Transputer Interface Module, the target and noise data files are transferred directly to the processors of the Seeker Scene Emulator at rates of 800 kilobytes per second.

This same MicroVax is used as the primary means of transferring programs and data to and from other contractors. Programs written for Vaxes and other off-site computers may be loaded onto this MicroVax via its nine-track tape drive. From there, files may be transferred to the PFP hosts (Intel 310s or Sun 386i's) or to other computer systems. Also, additional simulation support is available on this system through the MatrixX and ACSL languages. Both languages provide an environment for the simulation of discrete and continuous-time systems, including a choice of integration methods. MatrixX also has a graphical user interface for entering simulation specifics. This MicroVax is approved for classified data processing.

Another MicroVax is dedicated to a Chromatics 3-D graphics workstation. This combination of machines may be directly connected to a PFP processor in order to display complex three-dimensional graphics during simulations. Both of these machines are approved for classified data processing.

2.4. Secure laboratory

The principal elements of the Digital Emulation Technology Laboratory are housed in a laboratory on the third floor of the Centennial Research Building which has been approved for classified operation up to the secret level. Within this facility are most of the machines which have been described, including:

- the 80386-based PFP, with FPP capability,
- two PFP host machines (Intel 310s),
- the Seeker Scene Emulator,
- the MicroVax with 9-track and EXABYTE tape drives,
- the MicroVax/Chromatics system, and
- an IBM-compatible PC, primarily for classified word processing.

Each of these machines is approved for classified processing. The two PFP host machines are functionally identical, with one always available as a backup. A safe is also provided for storage of classified documents and magnetic media. All hard disks on classified machines are removable, and the classified operating disks are stored in the safe.

3. System Software

This chapter covers the latest changes which have been made to system software which supports the Floating-Point Processor (FPP) and the Crossbar/Sequencer subsystem. These changes represent incremental improvements and corrections that were required to make these tools operate within the current PFP. Not covered here are the completely new versions of system software which were developed for the new RMXII-based host. This effort is documented in the other DETL final report for this past year.

3.1. Floating-Point Processor

Significant updates have been made to both the Floating-Point Processor (FPP) Loader and the FPP Compiler, running under the iRMX II operating system. These are described in the next two sections.

3.1.1. Compiler

The PFP floating point processor compiler supports a subset of Pascal. Data types supported include 32 bit real, 32 bit integer, and arrays. The compiler recognizes FOR-DO, WHILE-DO, IF-THEN-ELSE, and BEGIN-END constructs. Any arbitrary arithmetic and boolean expressions are allowed. In addition, the compiler supports procedures and functions with arbitrary number of call-by-value and call-by-reference parameters.

The Pascal compiler was designed to produce efficient code for the PFP floating point processor without the generation of any intermediate code. Instead, the compiler directly maps the high level language expressions into executable machine code.

The Pascal compiler was originally written in Pascal and tested on a PC/AT system. The operating system dependent portions of the Pascal compiler were rewritten and then recompiled

and tested on the iRMX II system. The source code for the compiler may be found in Appendix B.

3.1.2. Loader

The PFP floating point processor loader supports the output of the Pascal compiler. The loader takes the output of the Pascal compiler and downloads code and data into the target floating point processor. The loader requires six steps:

1. Download a bootstrap program.

2. Start the bootstrap program.

3. Send application program data to the bootstrap program.

4. Stop the bootstrap program.

5. Download the application program code.

6. Start the application program.

The loader was originally written in Pascal and tested on a PC/AT system. The loader was rewritten in C and then recompiled and tested for the iRMX II system. The source code for the loader may be found in Appendix C.

3.2. Crossbar and Sequencer Compiler

In order to execute a parallel program on the PFP, crossbar and sequencer code is needed to describe the required communications. The compiler reads a simple language which describes the communications and generates three output files. One output file contains the communications and indicates the condition of the status lights during a simulation, this is useful for debugging. If the compiler detects an error in the input file, an error message will be placed in this output file and the compiler will stop. The remaining two output files contains the absolute code for the sequencer and crossbar for the particular simulation. The sequencer absolute code is generated as the compiler reads the input file, whereas the crossbar absolute code

is generated after the compiler has read the entire input file. To use the crossbar memory efficiently the same switch pattern can repeatedly be used, thus reducing the crossbar memory requirements. This is accomplished by reusing the next crossbar address in the sequencer code. These two absolute files can be directly loaded to the sequencer and crossbar without any intermediate steps.

The compiler was originally written in Pascal and tested on a iRMX I hosted PFP system. The compiler was modified and then recompiled and tested on a iRMX II hosted PFP system. Modifications include changing the absolute code format to conform to the iRMX II absolute code format. Also, the language used to describe the communications was enhanced so as to allow for two distinct crossbars and sequencers per host system. The source code for the compiler may be found in Appendix D.

4. Application Software

During the past contract year, most of the programming effort has focused on system software, but some major developments have also occurred with application software. This chapter describes both the Spinning Missile simulation and the preliminary KWEST/EXOSIM activity.

4.1. Spinning Missile

The Spinning Missile problem is a benchmark used by Electronic Associates, Inc. It had already been implemented on earlier versions of the PFP, but it was recently ported to the 286-based and 386-based PFPs with their new complement of system software. The problem is a six-degree-of-freedom missile represented by eighteen states. The equations of motion for the missile are given in Figure 4.1.

$$u'_s = r_s v_s - q_s w_s + (1/m)[T - 1/2p(u_s + W_x)^2 ACD_o] - g\sin O'$$

$$v'_s = -r_s u_s + (1/m)[-1/2pu_s AC_{Na}(v_s - W_y) + F_{TY}]$$

$$w'_s = q_s u_s + (1/m)[-1/2pu_s AC_{Na}(w_s + W_z) - F_{TZ}] + g\cos O'$$

$$p'_s = (1/I_{xs})[-1/4pu_s AD^2 C_{LP} p_s + 1/2pu_s^{\,2} ADC_{Ldt} dt + L_T + D_x p_s]$$

$$q'_s = (1/I_{ys})[1/2pu_s ADC_{ma}(w_s + W_z) + 1/4pu_s AD^2 C_{mq} q_s$$
$$-(L_C - L_{CG})F_{TZ} - r_s p_s I_{xs}]$$

$$r'_s = (1/I_{ys})[-1/2pu_s ADC_{ma}(v_s - W_y) + 1/4pu_s AD^2 C_{mq} r_s$$
$$-(L_C - L_{CG})F_{TY} + p_s q_s I_{xs}]$$

Figure 4.1 Spinning Missile equations of motion

The missile model was originally programmed in ACSL on a PC, to have a standard to compare the PFP results with. The problem was then partitioned onto thirty-two processors in the PFP. Eighteen of the processors are used to integrate the states and the remaining fourteen are used to calculate table values based on time or state values. These tables represent atmospheric density, wind coordinates, aerodynamic coefficients, spin torque, thrust, control force moment arm, missile mass and moment of inertia. All interpolation is linear.

The PFP with 80286 based processors (Intel iSBC 286/12) were used to time the simulation. For a 0.5 millisecond integration step size (fourth order Runge-Kutta), the PFP takes 10.23 milliseconds for an equivalent time step. Real-time performance was not expected using the 286/12 processor. Analysis of computational and communication loads for this problem yields real-time performance if the GT-FPP/3 (high speed processor) is substituted for the 286/12 processor.

The missile model, which was originally partitioned into thirty-two Pascal programs and tested on a iRMX I hosted PFP system, was modified and then recompiled on a iRMX II hosted PFP system. The missile simulation was then rerun as verification that the iRMX II system and its associated development tools produced the same results as the iRMX I system. The source code for the missile model may be found in Appendix E.

4.2. KWEST/EXOSIM

As noted earlier, the implementation of EXOSIM is an ongoing effort and is described in a separate final report. In the context of this contract, EXOSIM is the culmination of a series of exoatmospheric simulations, as shown in Figure 4.2. In this section we will provide a brief overview of the activity which has led up to the current project in which we are attempting to fully parallelize EXOSIM.

Figure 4.2: Evolution of EXOSIM

ERIS Baseline Specifications – LMSC

KWEST Simulation – BDM
 ACSL/FORTRAN

KEERIS Simulation – CRC (10/88-2/89)
 Boost-phase only
 ACSL/FORTRAN

EXOSIM Version 1.0 Simulation – CRC (3/89-6/89)
 Post-boost, midcourse, KV phases modeled
 All-FORTRAN
 BDM staring FPA seeker

EXOSIM Version 2.0 Simulation – CRC (7/89-10/89)
 Enhanced seeker, IMU
 SP/OP algorithms added
 Modifications to midcourse guidance and attitude control
 All-FORTRAN

Unclassified EXOSIM – Dynetics (1/90-5/90)
 Based on Version 1.0
 First- and second-stage boost only
 Time-driven, not event-driven
 Commented for parallel partitioning (up to five processors)

Parallel EXOSIM – Georgia Tech (3/90-present)
 Based on Unclassified EXOSIM
 Partitioned for up to 27 processors

The KWEST simulation is still being used indirectly at DETL. The data files used by the Seeker Scene Emulator are generated partly by a version of KWEST running at BDM. This activity is described in the final report on the Seeker Scene Emulator. Also, KWEST is being used as a basis for a 6DOF simulation written entirely in C, with the intent of porting it to a PFP populated with Georgia Tech Floating-Point Processors. This activity, too, is described elsewhere.

As we considered a candidate simulation for the PFP, Version 1.0 of EXOSIM was not considered promising for parallel implementation because of its inherent event-driven structure. We received some assistance from a subcontractor (Dynetics) in converting EXOSIM 1.0 to an unclassified, time-driven version. Although this unclassified EXOSIM was developed and tested on a single-processor system, parallel partitions were identified with inline comments. These partitions were tested by shuffling them within the main loop of the program.

The result of this was that it was relatively easy to port the simulation to the PFP (about 1 day). There were some uninitialized variables and a few minor deviations from ANSI FORTRAN (excessive continuation lines, do loops, and nonstandard initialization of common-block variables), but the basic porting process was reasonably straightforward, unlike many earlier simulations which required a great deal of manual effort.

The current status of EXOSIM is given in Figure 4.3.

Figure 4.3: Current Status of EXOSIM at DETL

Dynetics versions
- 4-processor and 5-processor versions both running on PFP
- Demonstrated feasibility of expressing parallelism in commented single-processor code
- Pre-tested code ported easily to PFP
- Did not fully exploit parallelism

GT parallel versions
- Developed in stages by restructuring, offloading table lookups, optimizing communication timing, predicting values, and minimizing double precision
- Running first on KDEC (286-based) PFP, then recently ported to development (386-based) PFP
- Further improvements will concentrate on minimizing communication time and achieving real-time performance

5. Appendices

A. Monitor source code


```
custom: do;

declare interrupt_number word public;

custom: procedure public;
          interrupt_number = 0;

          call setw( 00000H, build$ptr( selector( 00000H ), 01000H ), 07800H );
          call setw( 00000H, build$ptr( selector( 01000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 02000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 03000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 04000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 05000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 06000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 07000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 08000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 09000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 0a000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 0b000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 0c000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 0d000H ), 00000H ), 08000H );
          call setw( 00000H, build$ptr( selector( 0e000H ), 00000H ), 08000H );

          do while ( interrupt_number = 0 );
          end;

          if ( interrupt_number = 32 )
          then
                    cause$interrupt( 32 );

          cause$interrupt( 3 );
end custom;

end custom;
```

B. Floating-Point Compiler source code


```
File: ADDR_GEN.PAS
program addr_gen;
const
   mbus_window = $d000;
   multibus_segment = $09;

var
   program_counter : integer;
   branch_opcode, am2910_opcode, branch_address : integer;
   infile : text;
   infilename : string[52];
   memory_bank, segment, i, offset : integer;
   word  : array[0..5] of integer;

begin
   segment := mbus_window;
   writeln ('  -Multibus adapter card is set to 64k window at $0D0000 ');
   offsett := port[$200];
   writeln ('   -multibus page register is set st IO address $202');
   port[$200] := multibus_segment;
   writeln ('   -Multibus page is set at $09 where fpp memory is mapped');
   memw[segment:$f000] := 0;
   writeln('   - AMD processor is halted');
   branch_opcode := $c;
   am2910_opcode := $0
   writeln ('input file : '); readln(infilrname);
   if infilename = '' then infilename := 'branchop.in';
   reset(infile, infilename);
   readln (unfile);
   program_counter := 0;
   while not eof (infile) do
      begin
         readln (infile, program_counter, am2910_opcode, branch_address);
         branch_address := (branch_address and $ccc) +
                           (word_shr(word_and (branch_address, $002 ), 1)) +
                           (word_shl(word_and (branch_address, $001 ), 1)) +
                           (word_shr(word_and (branch_address, $020 ), 1)) +
                           (word_shl(word_and (branch_address, $010 ), 1)) +
```

```
                    (word_shr(word_and (branch_address, $200 ), 1)) +
                    (word_shl(word_and (branch_address, $100 ), 1));
      word[4] := word_shl (branch_opcode, 12);
      word[5] := word_shl (am2910_opcode, 12) + (branch_address);
      for memory_bank := 4 to 5 do
        memw[word_or (segment, word_shl (memory_bank, 9):
        word_shl (program_counter,1)] := word[memory_bank];
    end;
writeln (;address generator program is down loaded.');
end.
```

```
File: ARITH.DEF
public arith;
  Procedure create_index_term;
  Procedure index_expression(evaluate:boolean);
  Procedure index_constant_term(evaluate : boolean);
  Procedure index_integer(evaluate : boolean);
  Procedure Boolean_expression;
  Procedure arithmetic_term;
  procedure standard_function_call(var fx : operand_type);
  Procedure unary_arithmetic_expression;
  Procedure arithmetic_constant;
  Procedure simple_arithmetic_expression;
  Procedure arithmetic_expression;
  Procedure compound_arithmetic_expression;
  Procedure index_assignment_statement;
  Procedure general_expression(head_operand : operand_type);
  Procedure assignment_statement;
  Procedure procedure_call;
  Procedure function_call(var fx : operand_type);
  Procedure fetch_parameter(var parameter : operand_type);
  Procedure fetch_assigned_parameter(parameter : operand_type);
  Procedure fetch_index(var index : operand_type);
  procedure fetch_operand(var operand : operand_type);
```

File: ARITH.PAS

```
module arith;
$include(arith.def)
$include(global.def)
$include(exprsion.def)
$include(exprtree.def)
$include(utility.def)
$include(fetch_tk.def)
$include(symbol_t.def)
$include(code_gen.def)
$include(declare.def)
$include(lib.def)
$include(emu_lib.def)
private arith;
Procedure create_index_term;
begin
  create_expression(new_expression);
  last_expression[expression_level]^.down := new_expression;
  new_expression^.up := last_expression[expression_level];
  last_expression[expression_level] := new_expression;
  assign_percent_variable(token);
  last_expression[expression_level]^.id := token;
  last_expression[expression_level]^.id_type := integer_symbol_type;
  last_expression[expression_level]^.operator := unary_index;
  create_expression(new_expression);
  last_expression[expression_level]^.down := new_expression;
  new_expression^.up := last_expression[expression_level];
  last_expression[expression_level] := new_expression;
end; { of create_index_term }

Procedure index_constant_term(evaluate : boolean);
var operand1 : token_type;
    operand1_value : longint;
    child_expression : expression_pointer;
    operand : operand_type;
begin
  operand1 := token;
  operand1_value := integer_constant_value;
  fetch_token;
  if token = multiply_token then
  begin
    child_expression := last_expression[expression_level];
    if not evaluate then last_expression[expression_level] := last_expression[expression_level]^.left;
    create_index_term;
    if operand1_value < 0 then
    begin
       token := blank_token;
       token[1] := '-';
       concat(token,operand1)
    end
      else token := operand1;
```

```
    symbol_type := integer_constant_symbol_type;
    find_symbol(token,symbol_type,symbol_value,found);
    if not found then
    begin
      insert_symbol(token,symbol_type,symbol_value);
      declare_constant(symbol_value,symbol_type,token);
    end;
    operand.id := token;
    operand.id_type := integer_symbol_type;
    operand.index := blank_token;
    operand.offset := 0;
    insert_sibling_expression(operand,null_operator);
    expression_operator[expression_level] := multiplication;
    fetch_token;
    compound_arithmetic_expression;
    delete_expression(last_expression[expression_level]);
    if evaluate then
    begin
      traverse_expression_tree;
    end
    else
    begin
      last_expression[expression_level] := child_expression;
    end;
  end
  else
  begin
    if evaluate then
      last_expression[expression_level]^.offset := last_expression[expression_level]^.offset + operand1_value
    else
      last_expression[expression_level]^.left^.offset := last_expression[expression_level]^.left^.offset +
operand1_value;
  end;
end; { of index_constant_term }

Procedure index_integer(evaluate : boolean);
label 1; { constant }
var child_expression : expression_pointer;
    operand1,operand2 : token_type;
    operand2_value : longint;
    operand2_symbol_type : longint;
    operator1 : token_type;
begin
  operand1 := token;
  fetch_token;
  if (token = plus_token) or (token = minus_token) then
  begin
    1 { constant }:
    operator1 := token;
    fetch_token;
    find_symbol(token,symbol_type,symbol_value,found);
```

```
if (not found) and (symbol_type = integer_constant_symbol_type) then
begin
  insert_symbol(token,symbol_type,symbol_value);
  declare_constant(symbol_value,symbol_type,token);
end;
operand2 := token;
operand2_value := integer_constant_value;
operand2_symbol_type := symbol_type;
if symbol_type = integer_constant_symbol_type then
begin
  fetch_token;
  if token = multiply_token then
  begin
    child_expression := last_expression[expression_level];
    if not evaluate then last_expression[expression_level] := last_expression[expression_level]^.left;
    create_index_term;
    operand.id := operand1;
    operand.id_type := integer_symbol_type;
    operand.index := blank_token;
    operand.offset := 0;
    insert_sibling_expression(operand,null_operator);
    if operator1 = plus_token then
      expression_operator[expression_level] := addition
    else
      expression_operator[expression_level] := subtraction;
    operand.id := operand2;
    operand.id_type := integer_constant_symbol_type;
    operand.index := blank_token;
    operand.offset := 0;
    insert_sibling_expression(operand,expression_operator[expression_level]);
    expression_operator[expression_level] := multiplication;
    fetch_token;
    compound_arithmetic_expression;
    delete_expression(last_expression[expression_level]);
    if evaluate then
    begin
      traverse_expression_tree;
    end
    else
    begin
      last_expression[expression_level] := child_expression;
    end;
  end
  else
  begin
    if evaluate then
    begin
      if operator1 = plus_token then
        last_expression[expression_level]^.offset := last_expression[expression_level]^.offset + operand2_value
      else
        last_expression[expression_level]^.offset := last_expression[expression_level]^.offset - operand2_value;
```

```
        end
        else
        begin
          if operator1 = plus_token then
            last_expression[expression_level]^.left^.offset :=
last_expression[expression_level]^.left^.offset+operand2_value
          else
            last_expression[expression_level]^.left^.offset := last_expression[expression_level]^.left^.offset-
operand2_value
;
        end;
        if (token = plus_token) or (token = minus_token) then goto 1;
        if token = close_square_bracket then
        begin
          if evaluate then
            last_expression[expression_level]^.index := operand1
          else
            last_expression[expression_level]^.left^.index := operand1;
        end;
      end
    end
    else
    if (symbol_type = integer_symbol_type) then
    begin
      operand2 := token;
      child_expression := last_expression[expression_level];
      if not evaluate then last_expression[expression_level] := last_expression[expression_level]^.left;
      create_index_term;
      operand.id := operand1;
      operand.id_type := integer_symbol_type;
      operand.index := blank_token;
      operand.offset := 0;
      insert_sibling_expression(operand,null_operator);
      token := operand2;
      symbol_type := integer_symbol_type;
      if operator1 = plus_token then
        expression_operator[expression_level] := addition
      else
        expression_operator[expression_level] := subtraction;
      compound_arithmetic_expression;
      delete_expression(last_expression[expression_level]);
      if evaluate then
      begin
        traverse_expression_tree;
      end
      else
      begin
        last_expression[expression_level] := child_expression;
      end;
    end
    else
```

```
    begin
      write_error('integer type expected                              ',token);
    end;
  end
  else
  if token = close_square_bracket then
  begin
    if evaluate then
     last_expression[expression_level]^.index := operand1
    else
      last_expression[expression_level]^.left^.index := operand1;
  end
  else
  if token = multiply_token then
  begin
    child_expression := last_expression[expression_level];
    if not evaluate then last_expression[expression_level] := last_expression[expression_level]^.left;
    create_index_term;
    operand.id := operand1;
    operand.id_type := integer_symbol_type;
    operand.index := blank_token;
    operand.offset := 0;
    insert_sibling_expression(operand,null_operator);
    expression_operator[expression_level] := multiplication;
    fetch_token;
    compound_arithmetic_expression;
    delete_expression(last_expression[expression_level]);
    if evaluate then
    begin
      traverse_expression_tree;
    end
    else
    begin
      last_expression[expression_level] := child_expression;
    end;
  end;
end; { of index_integer }

Procedure index_expression(evaluate:boolean);
label 1;
var operand1,operand2,operator1: token_type;
    operand1_value : longint;
    child_expression : expression_pointer;
    operand2_symbol_type : longint;
begin
  1 :
  constant_assignment_type := integer_constant_symbol_type;
  find_symbol(token,symbol_type,symbol_value,found);
  if symbol_type = integer_symbol_type then
  begin
    index_integer(evaluate);
```

```
end
else
if symbol_type = integer_constant_symbol_type then
begin
  index_constant_term(evaluate);
  if token <> close_square_bracket then
  begin
    if token = plus_token then fetch_token;
    goto 1 { start };
  end;
end
else
if token = plus_token then
begin
  fetch_token;
  goto 1 { start };
end
else
if token = minus_token then
begin
  fetch_token;
  find_symbol(token,symbol_type,symbol_value,found);
  if symbol_type = integer_constant_symbol_type then
  begin
    integer_constant_value := -integer_constant_value;
    index_constant_term(evaluate);
    if token <> close_square_bracket then
    begin
      if token = plus_token then fetch_token;
      goto 1; { start }
    end;
  end
  else
  if symbol_type = integer_symbol_type then
  begin
    operand1 := token;
    child_expression := last_expression[expression_level];
    if not evaluate then last_expression[expression_level] := last_expression[expression_level]^.left;
    create_expression(new_expression);
    last_expression[expression_level]^.down := new_expression;
    new_expression^.up := last_expression[expression_level];
    last_expression[expression_level] := new_expression;
    assign_percent_variable(token);
    last_expression[expression_level]^.id := token;
    last_expression[expression_level]^.operator := unary_index;
    create_expression(new_expression);
    last_expression[expression_level]^.down := new_expression;
    new_expression^.up := last_expression[expression_level];
    last_expression[expression_level] := new_expression;
    str_integer(0,token);
    operand.id := token;
```

```
      operand.id_type := integer_symbol_type;
      operand.index := blank_token;
      operand.offset := 0;
      insert_sibling_expression(operand,null_operator);
      expression_operator[expression_level] := subtraction;
      token := operand1;
      compound_arithmetic_expression;
      delete_expression(last_expression[expression_level]);
      if evaluate then
      begin
        traverse_expression_tree;
      end
      else
      begin
        last_expression[expression_level] := child_expression;
      end;
    end;
  end
  else
  if token = open_parenthesis then
  begin
    child_expression := last_expression[expression_level];
    if not evaluate then last_expression[expression_level] := last_expression[expression_level]^.left;
    create_expression(new_expression);
    last_expression[expression_level]^.down := new_expression;
    new_expression^.up := last_expression[expression_level];
    last_expression[expression_level] := new_expression;
    assign_percent_variable(token);
    last_expression[expression_level]^.id := token;
    last_expression[expression_level]^.operator := unary_index;
    create_expression(new_expression);
    last_expression[expression_level]^.down := new_expression;
    new_expression^.up := last_expression[expression_level];
    last_expression[expression_level] := new_expression;
    expression_operator[expression_level] := null_operator;
    token := open_parenthesis;
    compound_arithmetic_expression;
    delete_expression(last_expression[expression_level]);
    if evaluate then
    begin
      traverse_expression_tree;
    end
    else
    begin
      last_expression[expression_level] := child_expression;
    end;
  end
  else
  begin
    write_error('integer expression expected                        ',token);
  end;
```

```
  constant_assignment_type := real_constant_symbol_type;
end; { of index_expression }

Procedure Boolean_expression;
var temp_token:token_type;
    b_op : longint;
    head_operand : operand_type;
begin
  if (token = true_token) or (token = false_token) then
  begin
    if write_lookahead_buffer[1].id <> blank_token then generate_Nop;
    if token = true_token then
    begin
{     branch_lookahead_buffer[0] := no_branch;
      first_expression[expression_level]^.address[1] := program_counter-2;
      first_expression[expression_level]^.address[2] := 7fffH; }
    end
    else
    begin
      branch_lookahead_buffer[0] := unconditional;
      first_expression[expression_level]^.address[1] := -program_counter+2;
      first_expression[expression_level]^.address[2] := 7fffH;
      microcode_address := program_counter;
      output_microcode_field;
      program_counter := program_counter + 1;
    end;
    fetch_token;
  end
  else
  begin
    temp_token := token;
    percent_variable_counter := 0;
    reset_temp_variable_address;
    assign_temp_variable(token);
    head_operand.id := token;
    head_operand.id_type := boolean_symbol_type;
    head_operand.index := blank_token;
    head_operand.offset := 0;
    reset_last_expression(head_operand);
    create_expression(new_expression);
    last_expression[expression_level]^.down := new_expression;
    new_expression^.up := last_expression[expression_level];
    last_expression[expression_level] := new_expression;
    token := temp_token;
    compound_arithmetic_expression;
    delete_expression(last_expression[expression_level]);
    traverse_expression_tree;
  end;
end;  { of boolean_expression }

Procedure arithmetic_term;
```

```
var child_expression : expression_pointer;
    operand : operand_type;
begin
  assign_percent_variable(token);
  child_expression := last_expression[expression_level];
  operand.id := token;
  operand.id_type := symbol_type;
  operand.index := blank_token;
  operand.offset := 0;
  insert_child_expression(operand,expression_operator[expression_level]);
  expression_operator[expression_level] := null_operator;
  fetch_token;
  compound_arithmetic_expression;
  verify_token(token,close_parenthesis);
  delete_expression(last_expression[expression_level]);
  last_expression[expression_level] := child_expression^.right;
end; { of arithmetic_term }

Procedure standard_function_call(var fx : operand_type);
var temp_constant_assignment_type : longint;
    function_token : token_type;
    x : operand_type;
begin
  temp_constant_assignment_type := constant_assignment_type;
  function_token := token;
  fetch_token;
  verify_token(token,open_parenthesis);
  fetch_parameter(x);
  verify_token(token,close_parenthesis);
  assign_parameter(fx,real_symbol_type);
  if (function_token=trunc_token) then
  begin
    function_trunc(fx,x);
  end
  else
  if (function_token=round_token) then
  begin
    function_round(fx,x);
  end
  else
  if (function_token = exp_token) then
  begin
    function_exp(fx,x);
  end
  else
  if (function_token = ln_token) then
  begin
    function_ln(fx,x);
  end
  else
  if (function_token = sqrt_token) then
```

```
 begin
   function_sqrt(fx,x);
 end
 else
 if (function_token = sin_token) then
 begin
   function_sin(fx,x);
 end
 else
 if (function_token = cos_token) then
 begin
   function_cos(fx,x);
 end
 else
 if (function_token = tan_token) then
 begin
   function_tan(fx,x);
 end
 else
 if (function_token = asin_token) then
 begin
   function_asin(fx,x);
 end
 else
 if (function_token = acos_token) then
 begin
   function_acos(fx,x);
 end
 else
 if (function_token = atan_token) then
 begin
   function_atan(fx,x);
 end
 else
 begin
   write_error('unsupported function                              ',function_token);
 end;
 constant_assignment_type := temp_constant_assignment_type;
end; { standard_function_call }

Procedure unary_arithmetic_expression;
var child_expression : expression_pointer;
    operand : operand_type;
begin
  child_expression := nil;
  if (token=minus_token) then
  begin
    if expression_operator[expression_level] = subtraction then
      expression_operator[expression_level] := addition
    else
    begin
```

```
      child_expression := last_expression[expression_level];
      assign_percent_variable(token);
      operand.id := token;
      operand.id_type := symbol_type;
      operand.index := blank_token;
      operand.offset := 0;
      insert_child_expression(operand,expression_operator[expression_level]);
      expression_operator[expression_level] := unary_minus;
    end;
  end
  else
  if (token=plus_token) then
  begin
    {do nothing}
  end
  else
  if (token=not_token) then
  begin
    child_expression := last_expression[expression_level];
    assign_percent_variable(token);
    operand.id := token;
    operand.id_type := symbol_type;
    operand.index := blank_token;
    operand.offset := 0;
    insert_child_expression(operand,expression_operator[expression_level]);
    expression_operator[expression_level] := unary_not;
  end;
  fetch_token;
  simple_arithmetic_expression;
  if child_expression <> nil then
  begin
    delete_expression(last_expression[expression_level]);
    last_expression[expression_level] := child_expression^.right;
  end;
end; { of unary arithmetic operator }

Procedure arithmetic_constant;
var operand : operand_type;
begin
  find_symbol(token,symbol_type,symbol_value,found);
  if ((symbol_type = integer_constant_symbol_type) or
      (symbol_type = real_constant_symbol_type)) and
      (not found) then
  begin
    symbol_type := real_constant_symbol_type;
    val_real(token,real_constant_value,i);
    str_real(real_constant_value,token);
  end;
  operand.id := token;
  operand.id_type := symbol_type;
  operand.index := blank_token;
```

```
  operand.offset := 0;
  insert_sibling_expression(operand,expression_operator[expression_level]);
  expression_operator[expression_level] := null_operator;
  find_symbol(token,symbol_type,symbol_value,found);
  if not found then
  begin
    insert_symbol(token,symbol_type,symbol_value);
    declare_constant(symbol_value,symbol_type,token);
  end;
end; { of arithmetic_constant }

Procedure simple_arithmetic_expression;
var fx,x,operand,index_operand : operand_type;
begin
  find_symbol(token,symbol_type,symbol_value,found);
  if ((symbol_type = integer_constant_symbol_type) or
      (symbol_type = real_constant_symbol_type)) and
      (not found) then
  begin
    insert_symbol(token,symbol_type,symbol_value);
    declare_constant(symbol_value,symbol_type,token);
  end;
  if expression_operator[expression_level] = division then
  begin
    fetch_operand(x);
    assign_parameter(fx,real_symbol_type);
    generate_reciprocal(fx,x);
    expression_operator[expression_level] := multiplication;
    insert_sibling_expression(fx,expression_operator[expression_level]);
    expression_operator[expression_level] := null_operator;
  end
  else
  if token = open_parenthesis then
    arithmetic_term
  else
  if (symbol_type = standard_function_symbol_type) then
  begin
    standard_function_call(operand);
    insert_sibling_expression(operand,expression_operator[expression_level]);
    expression_operator[expression_level] := null_operator;
  end
  else
  if (symbol_type = function_symbol_type) then
  begin
    function_call(operand);
    insert_sibling_expression(operand,expression_operator[expression_level]);
    expression_operator[expression_level] := null_operator;
  end
  else
  if (symbol_type = real_symbol_type) or
    (symbol_type = integer_symbol_type) or
```

```
    (symbol_type = boolean_symbol_type) or
    (symbol_type = boolean_constant_symbol_type) then
  begin
    operand.id := token;
    operand.id_type := symbol_type;
    operand.index := blank_token;
    operand.offset := 0;
    insert_sibling_expression(operand,expression_operator[expression_level]);
    expression_operator[expression_level] := null_operator;
  end
  else
  if (symbol_type = real_constant_symbol_type) or
    (symbol_type = integer_constant_symbol_type) then
    arithmetic_constant
  else
  if (symbol_type = real_array_symbol_type) then
  begin
    symbol_type := real_symbol_type;
    operand.id := token;
    operand.id_type := symbol_type;
    fetch_token;
    verify_token(token,open_square_bracket);
    fetch_index(index_operand);
    operand.index := index_operand.id;
    operand.offset := index_operand.offset;
    verify_token(token,close_square_bracket);
    insert_sibling_expression(operand,expression_operator[expression_level]);
  end
  else
  if (symbol_type = integer_array_symbol_type) then
  begin
    symbol_type := integer_symbol_type;
    operand.id := token;
    operand.id_type := symbol_type;
    fetch_token;
    verify_token(token,open_square_bracket);
    fetch_index(index_operand);
    operand.index := index_operand.id;
    operand.offset := index_operand.offset;
    verify_token(token,close_square_bracket);
    insert_sibling_expression(operand,expression_operator[expression_level]);
  end
  else
  if (symbol_type = boolean_array_symbol_type) then
  begin
    symbol_type := boolean_symbol_type;
    operand.id := token;
    operand.id_type := symbol_type;
    fetch_token;
    verify_token(token,open_square_bracket);
    fetch_index(index_operand);
```

```
    operand.index := index_operand.id;
    operand.offset := index_operand.offset;
    verify_token(token,close_square_bracket);
    insert_sibling_expression(operand,expression_operator[expression_level]);
  end
  else
  begin
    write_error('invalid id                                        ',token);
  end;
end; { of simple arithmetic expression }

Procedure arithmetic_expression;
begin
  if (token = minus_token) or (token = plus_token) or
     (token = not_token) then
    unary_arithmetic_expression
  else
    simple_arithmetic_expression;
end; {of arithmetic_expression}

Procedure compound_arithmetic_expression;
begin
  arithmetic_expression;
  fetch_token;
  if (token=plus_token) then
  begin
    expression_operator[expression_level] := addition;
    fetch_token;
    compound_arithmetic_expression;
  end
  else
  if (token=minus_token) then
  begin
    expression_operator[expression_level] := subtraction;
    fetch_token;
    compound_arithmetic_expression;
  end
  else
  if (token=multiply_token) then
  begin
    expression_operator[expression_level] := multiplication;
    fetch_token;
    compound_arithmetic_expression;
  end
  else
  if (token=divide_token) then
  begin
    expression_operator[expression_level] := division;
    fetch_token;
    compound_arithmetic_expression;
  end
```

```
else
if (token=greater_than_token) then
begin
  expression_operator[expression_level] := greater_than;
  fetch_token;
  compound_arithmetic_expression;
end
else
if (token=greater_than_or_equal_token) then
begin
  expression_operator[expression_level] := greater_than_or_equal;
  fetch_token;
  compound_arithmetic_expression;
end
else
if (token=less_than_token) then
begin
  expression_operator[expression_level] := less_than;
  fetch_token;
  compound_arithmetic_expression;
end
else
if (token=less_than_or_equal_token) then
begin
  expression_operator[expression_level] := less_than_or_equal;
  fetch_token;
  compound_arithmetic_expression;
end
else
if (token=equal_token) then
begin
  expression_operator[expression_level] := equal;
  fetch_token;
  compound_arithmetic_expression;
end
else
if (token=not_equal_token) then
begin
  expression_operator[expression_level] := not_equal;
  fetch_token;
  compound_arithmetic_expression;
end
else
if (token=and_token) then
begin
  expression_operator[expression_level] := and_operation;
  fetch_token;
  compound_arithmetic_expression;
end
else
if (token= or_token) then
```

```
  begin
    expression_operator[expression_level] := or_operation;
    fetch_token;
    compound_arithmetic_expression;
  end;
end;   { of compound arithmetic expression }

Procedure index_assignment_statement;
var head_operand,index_operand : operand_type;
begin
{  writeln(outfile,';-------- expression ',expression_number,' --------'); }
  clear_temp_index;
  reset_temp_variable_address;
  expression_number := expression_number + 1;
  percent_variable_counter := 0;
  head_operand.id := token;
  head_operand.id_type := symbol_type;
  fetch_token;
  verify_token(token,open_square_bracket);
  fetch_index(index_operand);
  head_operand.index := index_operand.id;
  head_operand.offset := index_operand.offset;
  reset_last_expression(head_operand);
  expression_operator[expression_level] := null_operator;
  last_expression[expression_level] := first_expression[expression_level];
  verify_token(token,close_square_bracket);
  create_expression(new_expression);
  last_expression[expression_level]^.down := new_expression;
  new_expression^.up := last_expression[expression_level];
  last_expression[expression_level] := new_expression;
  fetch_token;
  verify_token(token,colon);
  fetch_token;
  verify_token(token,equal_token);
  fetch_token;
  compound_arithmetic_expression;
  delete_expression(last_expression[expression_level]);
  traverse_expression_tree;
end; { of index_assignment_statement }

Procedure general_expression(head_operand : operand_type);
begin
{  writeln(outfile,';-------- expression ',expression_number,' --------'); }
  expression_number := expression_number + 1;
  percent_variable_counter := 0;
  reset_last_expression(head_operand);
  create_expression(new_expression);
  last_expression[expression_level]^.down := new_expression;
  new_expression^.up := last_expression[expression_level];
  last_expression[expression_level] := new_expression;
  fetch_token;
```

```
  compound_arithmetic_expression;
  delete_expression(last_expression[expression_level]);
  transform_expression_tree;
end; { of general_expression }

Procedure assignment_statement;
var head_operand : operand_type;
begin
{   writeln(outfile,';-------- expression ',expression_number,' --------'); }
  expression_number := expression_number + 1;
  clear_temp_index;
  percent_variable_counter := 0;
  head_operand.id := token;
  head_operand.id_type := symbol_type;
  head_operand.index := blank_token;
  head_operand.offset := 0;
  reset_last_expression(head_operand);
  create_expression(new_expression);
  last_expression[expression_level]^.down := new_expression;
  new_expression^.up := last_expression[expression_level];
  last_expression[expression_level] := new_expression;
  fetch_token;
  verify_token(token,colon);
  fetch_token;
  verify_token(token,equal_token);
  fetch_token;
  reset_temp_variable_address;
  compound_arithmetic_expression;
  delete_expression(last_expression[expression_level]);
  traverse_expression_tree;
end; { of assignment_statement }

Procedure procedure_call;
var procedure_address : longint;
    reference_actual_parameter : array[0..max_reference_parameter] of operand_type;
    reference_formal_parameter : array[0..max_reference_parameter] of operand_type;
    i : longint;
    no_reference_parameter : longint; { use to keep track of the number of call by value parameters }
    actual_parameter,formal_parameter: operand_type;
begin
  for i := 0 to max_index_register do index_register[i] := blank_token;
  reset_temp_variable_address;
  no_reference_parameter := 0;
  current_parameter := procedure_link^.parameter_link;
  procedure_address := procedure_link^.value;
  if current_parameter <> nil then
  begin
    fetch_token;
    verify_token(token,open_parenthesis);
    while current_parameter <> nil do
    begin
```

```
        fetch_parameter(actual_parameter);
        symbol_type := actual_parameter.id_type;
        find_symbol(actual_parameter.id,symbol_type,symbol_value,found);
        expression_number := expression_number+1;
        simplify_type(symbol_type);
        if symbol_type <> current_parameter^.id_type then
          write_error('Type mismatch :                              ',actual_parameter.id);
        str_integer(current_parameter^.address,formal_parameter.id);
        temp_token := blank_token;
        temp_token[1] := '#';
        concat(temp_token, formal_parameter.id);
        formal_parameter.id := temp_token;
        formal_parameter.id_type := current_parameter^.id_type;
        formal_parameter.offset := 0;
        formal_parameter.index := blank_token;
        generate_ALU_operation(formal_parameter,actual_parameter,zero_operand,addition);
        if current_parameter^.parameter_type = call_by_reference then
        begin
          no_reference_parameter := no_reference_parameter+1;
          if actual_parameter.id[1] = '#' then
            write_error('call by reference parameter is expected          ',blank_token);
          reference_actual_parameter[no_reference_parameter] := actual_parameter;
          reference_formal_parameter[no_reference_parameter] := formal_parameter;
        end;
        current_parameter := current_parameter^.next;
        if current_parameter <> nil then verify_token(token,comma);
      end;
      verify_token(token,close_parenthesis);
    end;
    if (write_lookahead_buffer[1].id <> blank_token) then generate_Nop;
    if (branch_lookahead_buffer[0] <> no_branch) then generate_Nop;
  writeln(outfile,';',program_counter,': gosub ',procedure_address);
    microcode_address := program_counter;
    branch_address := procedure_address;
    AM2910_opcode := CJS;
    branch_opcode := unconditional;
    output_microcode_field;
    program_counter := program_counter + 1;
    for i := 1 to no_reference_parameter do
    begin
      generate_ALU_operation(reference_actual_parameter[i],reference_formal_parameter[i],zero_operand,addition);
    end;
    for i := 0 to max_index_register do index_register[i] := blank_token;
    fetch_token;
  end; { of procedure_call }

  Procedure function_call(var fx : operand_type);
  var procedure_address : longint;
      reference_actual_parameter : array[0..max_reference_parameter] of operand_type;
      reference_formal_parameter : array[0..max_reference_parameter] of operand_type;
      i : longint;
```

```
    no_reference_parameter : longint; { use to keep track of the number of call by value parameters }
    actual_parameter,formal_parameter: operand_type;
    function_operator : longint;
    x : operand_type;
    current_parameter : parameter_pointer;
begin
  for i := 0 to max_index_register do index_register[i] := blank_token;
  function_operator := expression_operator[expression_level];
  no_reference_parameter := 0;
  current_parameter := procedure_link^.parameter_link;
  str_integer(current_parameter^.address,fx.id);
  temp_token := blank_token;
  temp_token[1] := '&';
  concat (temp_token, fx.id);
  fx.id := temp_token;
  fx.id_type := current_parameter^.id_type;
  fx.index := blank_token;
  fx.offset := 0;
  current_parameter := current_parameter^.next;
  procedure_address := procedure_link^.value;
  if current_parameter <> nil then
  begin
    fetch_token;
    verify_token(token,open_parenthesis);
    while current_parameter <> nil do
    begin
      fetch_parameter(actual_parameter);
      symbol_type := actual_parameter.id_type;
      find_symbol(actual_parameter.id,symbol_type,symbol_value,found);
      expression_number := expression_number+1;
      simplify_type(symbol_type);
      if symbol_type <> current_parameter^.id_type then
        write_error('Type mismatch :                                    ',actual_parameter.id);
      str_integer(current_parameter^.address,formal_parameter.id);
      temp_token := blank_token;
      temp_token[1] := '#';
      concat(temp_token, formal_parameter.id);
      formal_parameter.id := temp_token;
      formal_parameter.id_type := current_parameter^.id_type;
      formal_parameter.offset := 0;
      formal_parameter.index := blank_token;
      generate_ALU_operation(formal_parameter,actual_parameter,zero_operand,addition);
      if current_parameter^.parameter_type = call_by_reference then
      begin
        no_reference_parameter := no_reference_parameter+1;
        if actual_parameter.id[1] = '#' then
          write_error('call by reference parameter is expected          ',blank_token);
        reference_actual_parameter[no_reference_parameter] := actual_parameter;
        reference_formal_parameter[no_reference_parameter] := formal_parameter;
      end;
      current_parameter := current_parameter^.next;
```

```
      if current_parameter <> nil then verify_token(token,comma);
    end;
    verify_token(token,close_parenthesis);
  end;
  if (write_lookahead_buffer[1].id <> blank_token) then generate_Nop;
  if (branch_lookahead_buffer[0] <> no_branch) then generate_Nop;
  writeln(outfile,';',program_counter,': gosub ',procedure_address);
  microcode_address := program_counter;
  branch_address := procedure_address;
  AM2910_opcode := CJS;
  branch_opcode := unconditional;
  output_microcode_field;
  program_counter := program_counter + 1;
  for i := 1 to no_reference_parameter do
  begin
    generate_ALU_operation(reference_actual_parameter[i],reference_formal_parameter[i],zero_operand,addition);
  end;
  x := fx;
  assign_parameter(fx,fx.id_type);
  generate_ALU_operation(fx,x,zero_operand,addition);
  for i := 0 to max_index_register do index_register[i] := blank_token;
  expression_operator[expression_level] := function_operator;
  for i := 0 to max_index_register do index_register[i] := blank_token;
end; { of function_call }

Procedure fetch_parameter(var parameter : operand_type);
var  found : boolean;
begin
  expression_level := expression_level+1;
  create_expression(new_expression);
  first_expression[expression_level] := new_expression;
{  assign_stack_operand(parameter,general_symbol_type); }
  assign_dummy_parameter(parameter);
  constant_assignment_type := real_symbol_type;
  reset_temp_variable_address;
  general_expression(parameter);
  current_expression := first_expression[expression_level];
{  node_display(current_expression);
  node_display(current_expression^.down); }
  if assignment_necessary(current_expression) then
  begin
    assign_parameter(parameter,parameter.id_type);
    current_expression^.id := parameter.id;
    current_expression^.id_type := parameter.id_Type;
    evaluate_expression_tree;
    fetch_expression_operand(current_expression,parameter);
  end
  else
  begin
    fetch_expression_operand(current_expression^.down,parameter);
    dispose(current_expression^.down^.right);
```

```
    dispose(current_expression^.down);
{    free_stack_operand; }
  end;
  dispose(first_expression[expression_level]);
  expression_level := expression_level-1;
end; { of fetch_parameter }

Procedure fetch_assigned_parameter(parameter : operand_type);
{ to fetch a parameter that has been allocated a location in the
  data memory }
var  found : boolean;
begin
  expression_level := expression_level+1;
  create_expression(new_expression);
  first_expression[expression_level] := new_expression;
  constant_assignment_type := real_symbol_type;
  reset_temp_variable_address;
  general_expression(parameter);
  current_expression := first_expression[expression_level];
{  node_display(current_expression);
  node_display(current_expression^.down); }
  evaluate_expression_tree;
  fetch_expression_operand(current_expression,parameter);
  dispose(first_expression[expression_level]);
  expression_level := expression_level -1;
end; { of fetch_assigned_parameter }

Procedure fetch_index(var index : operand_type);
var  found : boolean;
     i : longint;
     constant_offset : longint;
begin
  expression_level := expression_level+1;
  create_expression(new_expression);
  first_expression[expression_level] := new_expression;
  assign_dummy_parameter(index);
  index.id_type := integer_symbol_type;
  constant_assignment_type := integer_symbol_type;
  general_expression(index);
  current_expression := first_expression[expression_level];
{  node_display(current_expression);
  node_display(current_expression^.down);
  node_display(current_expression^.down^.right); }
  if index_assignment_necessary(current_expression) then
  begin
    assign_parameter(index,index.id_type);
    current_expression^.id := index.id;
    current_expression^.id_type := index.id_type;
    evaluate_expression_tree;
    fetch_expression_operand(current_expression,index);
  end
```

```
  else
  begin
    val_integer(current_expression^.down^.right^.id,constant_offset,i);
    fetch_expression_operand(current_expression^.down,index);
    if current_expression^.down^.right^.operator = subtraction then
      index.offset := index.offset - constant_offset
    else
      index.offset := index.offset + constant_offset;
    dispose(current_expression^.down^.right);
    dispose(current_expression^.down);
{    free_stack_operand; }
  end;
  dispose(first_expression[expression_level]);
  expression_level := expression_level -1;
end; { of fetch_index }

procedure fetch_operand(var operand : operand_type);
var index_operand : operand_type;
begin
  reset_operand(operand);
  if token = open_parenthesis then
  begin
    fetch_parameter(operand);
    symbol_type := operand.id_type;
  end
  else
  if (symbol_type = standard_function_symbol_type) then
  begin
    standard_function_call(operand);
    symbol_type := operand.id_type;
  end
  else
  if (symbol_type = function_symbol_type) then
  begin
    function_call(operand);
    symbol_type := operand.id_type;
  end
  else
  if (symbol_type = real_symbol_type) or
    (symbol_type = integer_symbol_type) or
    (symbol_type = real_constant_symbol_type) or
   (s%[ PrinterErrymbol_type = integer_constant_symbol_type) then
  begin
    operand.id := token;
    operand.id_type := symbol_type;
    operand.index := blank_token;
    operand.offset := 0;
  end
  else
  if (symbol_type = real_array_symbol_type) then
  begin
```

```
    symbol_type := real_symbol_type;
    operand.id := token;
    operand.id_type := symbol_type;
    fetch_token;
    verify_token(token,open_square_bracket);
    fetch_index(index_operand);
    operand.index := index_operand.id;
    operand.offset := index_operand.offset;
    verify_token(token,close_square_bracket);
  end
  else
  if (symbol_type = integer_array_symbol_type) then
  begin
    symbol_type := integer_symbol_type;
    operand.id := token;
    operand.id_type := symbol_type;
    fetch_token;
    verify_token(token,open_square_bracket);
    fetch_index(index_operand);
    operand.index := index_operand.id;
    operand.offset := index_operand.offset;
    verify_token(token,close_square_bracket);
  end
  else
  if (symbol_type = boolean_array_symbol_type) then
  begin
    symbol_type := boolean_symbol_type;
    operand.id := token;
    operand.id_type := symbol_type;
    fetch_token;
    verify_token(token,open_square_bracket);
    fetch_index(index_operand);
    operand.index := index_operand.id;
    operand.offset := index_operand.offset;
    verify_token(token,close_square_bracket);
  end
  else
  begin
    write_error('invalid id                                   ',token);
  end;
end;.
```

```
File: BIT_FUNC.DEF
public bit_func;
  function word_shr(value,num:word):word;
  function word_shl(value,num:word):word;
  function word_and(value,num:word):word;
  function word_or(value,num:word):word;
  function word_xor(value,num:word):word;
  function dword_shr(value,num:word):longint;
  function dword_shl(value,num:word):longint;
  function dword_and(value,num:word):longint;
  function dword_or(value,num:word):longint;
  function dword_xor(value,num:word):longint;
```

```
File: BIT_FUNC.PLM
bit_funcs: do;


word_or : procedure ( value1, value2 ) word public;

    declare ( value1, value2 ) word;

    return (value1 or value2);

end word_or;

word_and : procedure ( value1, value2 ) word public;

    declare ( value1, value2 ) word;

    return (value1 and value2);

end word_and;

word_shl : procedure ( value1, count ) word public;

    declare ( value1, count ) word;

    return ( shl(value1,count) );

end word_shl;

word_shr : procedure ( value1, count ) word public;

    declare ( value1, count ) word;

    return ( shr(value1,count) );

end word_shr;

word_xor : procedure ( value1, value2 ) word public;

    declare ( value1, value2 ) word;

    return ( value1 xor value2 );

end word_xor;


dword_or : procedure ( value1, value2 ) dword public;

    declare ( value1, value2 ) word;

    return (value1 or value2);
```

```
end dword_or;

dword_and : procedure ( value1, value2 ) dword public;

    declare ( value1, value2 ) word;

    return (value1 and value2);

end dword_and;

dword_shl : procedure ( value1, count ) dword public;

    declare ( value1, count ) word;

    return ( shl(value1,count) );

end dword_shl;

dword_shr : procedure ( value1, count ) dword public;

    declare ( value1, count ) word;

    return ( shr(value1,count) );

end dword_shr;

dword_xor : procedure ( value1, value2 ) dword public;

    declare ( value1, value2 ) word;

    return ( value1 xor value2 );

end dword_xor;


end bit_funcs;
```

File: CODE_GEN.DEF

```
public code_gen;
  procedure swap_operand(var R,S:operand_type);
  Procedure reset_microcode_field;
  Procedure display_branch_condition;
  Procedure output_microcode_field;
  Procedure generate_Nop;
  Procedure display_ALU_operation(F,R,S:operand_type;opcode:longint);
  Procedure Bind_ALU_opcode(opcode : longint);
  Procedure Bind_AF_AR_AS(F,R,S:operand_type);
  procedure bind_AIF_AIR_AIS(F,R,S:operand_type);
  Procedure generate_ALU_operation(F,R,S:operand_type;opcode:longint);
  Procedure Clear_pipeline_stage;
  Procedure check_F_bus(operand : operand_type);
  Procedure load_index_register(index : token_Type);
  Procedure check_index_register(var F,R,S : operand_type);
  Procedure check_pipeline_stage(var F,R,S:operand_type;var opcode :longint);
  Procedure find_branch_address(var expression : expression_pointer;
                                                  var branch_state : longint);
  procedure generate_boolean_assignment_microcode(var
                                            expression : expression_pointer;
                                            branch_true : boolean);
  Procedure generate_branch_address(var expression : expression_pointer);
  procedure assign_temp_boolean_variable(var F : operand_type);
  procedure generate_greater_than(var expression : expression_pointer);
  procedure generate_less_than(var expression : expression_pointer);
  procedure generate_equal(var expression : expression_pointer);
  procedure generate_not_equal(var expression : expression_pointer);
  procedure generate_greater_than_or_equal(var
                                             expression : expression_pointer);
  procedure generate_less_than_or_equal(var expression : expression_pointer);
  Procedure generate_unary_not(var expression : expression_pointer);
  procedure generate_and(var expression : expression_pointer);
  procedure generate_or(var expression : expression_pointer);
  function assign_index(index : token_type):longint;
  Procedure generate_read_function(function_number : longint;fx,x : operand_type);
  Procedure assign_R_bus(operand:operand_type);
  Procedure assign_S_bus(operand:operand_type);
  Procedure assign_F_bus(operand:operand_type);
```

```
File: CODE_GEN.PAS
module code_gen;
$include(code_gen.def)
$include(global.def)
$include(utility.def)
$include(symbol_t.def)
$include(emu_lib.def)

private code_gen;
Procedure assign_R_bus(operand:operand_type);
begin
  AR := operand.id_address + operand.offset;
  if (operand.index <> blank_token) and (operand.id[1] <> '#') and
     (operand.index[1] <> '0') and (operand.id[1] <> '&') then
  begin
    AIR[0] := operand.index_address;
    IA1 := 1;
  end;
end; { of assign_R_bus }

Procedure assign_S_bus(operand:operand_type);
begin
  AS := operand.id_address + operand.offset;
  if (operand.index <> blank_token) and (operand.id[1] <> '#') and
     (operand.index[1] <> '0') and (operand.id[1] <> '&') then
  begin
    AIS[0] := operand.index_address;
    IA0 := 1;
  end;
end; { of assign_S_bus }

Procedure assign_F_bus(operand:operand_type);
begin
  AF[0] := operand.id_address + operand.offset;
  if (operand.index <> blank_token) and (operand.id[1] <> '#') and
     (operand.index[1] <> '0') and (operand.id[1] <> '&') then
  begin
    AIF[0] := operand.index_address;
    IA2[0] := 1;
  end;
end; { of assign_F_bus }

Procedure reset_microcode_field;
var i : longint;
begin
  am2910_opcode := cont;
  branch_address := 0;
  branch_opcode := no_branch;
  for i := 1 to 2 do
  begin
    write_lookahead_buffer[i-1] := write_lookahead_buffer[i];
```

```
    AF[i-1] := AF[i];
    branch_lookahead_buffer[i-1] := branch_lookahead_buffer[i];
    AIF[i-1] := AIF[i];
    IA2[i-1] := IA2[i];
  end;
  AIR[0] := AIR[1];
  AIS[0] := AIS[1];
  AIR[1] := 7fffH;
  AIS[1] := 7fffH;
  AIF[2] := 7fffH;
  with write_lookahead_buffer[2] do
  begin
    id := blank_token;
    index := blank_token;
    offset := 0;
  end;
  branch_lookahead_buffer[2] := 7fffH;
  AF[2] := 0;
  mc325_buffer[0] := mc325_buffer[1];
  mc325_buffer[1] := 0;
  I4 := 0;
  ENF_bar := 0;
  Dsel := 0;
  I3[0] := I3[1];
  I3[1] := 0;
  msw := 0;
  read_opcode := 0;
  IA2[2] := 0;
  IA1 := 0;
  IA0 := 0;
  AR := 1;
  AS := 1;
  write_opcode := 0;
end;

Procedure display_branch_condition;
begin
  if branch_opcode = 0 then
    writeln(outfile,';',microcode_address,': branch if Network FIFO has no data')
  else
  if branch_opcode = 1 then
    writeln(outfile,';',microcode_address,': branch if Network FIFO is not ready for input')
  else
  if branch_opcode = 2 then
    writeln(outfile,';',microcode_address,': branch if Host FIFO has no data')
  else
  if branch_opcode = 3 then
    writeln(outfile,';',microcode_address,': branch if Host FIFO is not ready for input')
  else
  if branch_opcode = 4 then
    writeln(outfile,';',microcode_address,': branch if zero')
```

```
  else
  if branch_opcode = 5 then
    writeln(outfile,';',microcode_address,': branch if not zero')
  else
  if branch_opcode = 6 then
    writeln(outfile,';',microcode_address,': branch if negatif')
  else
  if branch_opcode = 7 then
  begin
    am2910_opcode := cont;
    writeln(outfile,': Nop');
  end
  else
  if branch_opcode = 9 then
    writeln(outfile,';',microcode_address,': branch if not negatif')
  else
  if branch_opcode = 0AH then
    writeln(outfile,';',microcode_address,': branch if error ')
  else
  if branch_opcode = 0CH then
    writeln(outfile,';',microcode_address,': unconditional branch')
  else
  begin
    writeln (errorfile);
    writeln (errorfile, 'unknown branch opcode : ',branch_opcode);
    error_found
  end
end;


Procedure output_microcode_field;
begin
  if AIF[0] = 7fffH then AIF[0] := 0;
  if AIR[0] = 7fffH then AIR[0] := 0;
  if AIS[0] = 7fffH then AIS[0] := 0;
  if write_opcode = 0 then
  begin
    if (write_lookahead_buffer[0].id <> blank_token) then
         write_opcode := 1;
  end;
  if branch_lookahead_buffer[0] <> 7fffH then
  begin
    am2910_opcode := CJP;
    branch_opcode := branch_lookahead_buffer[0];
    display_branch_condition;
  end;
  write(outfile,'c ',microcode_address:5);
  write(outfile,am2910_opcode:3,branch_address:5,branch_opcode:3,write_opcode:3);
  write(outfile,Dsel:2,read_opcode:2,ENF_bar:2,I4:2,I3[0]:2,MC325_buffer[0]:3);
  writeln(outfile,AF[0]:5,AR:5,AS:5,msw:2,IA2[0]:2,IA1:2,IA0:2,AIF[0]:3,AIR[0]:3,AIS[0]:3);
  reset_microcode_field;
```

```
  if microcode_address > program_counter_limit then
  begin
    writeln;
    writeln(errorfile,'Microcode address exceeds ',program_counter_limit);
    writeln(errorfile,'Try to partition this process into two processes. ');
    error_found;
  end;
end; { of output microcode_field }

Procedure generate_Nop;
begin
  microcode_address := program_counter;
  writeln(outfile,';',program_counter,': Nop');
  output_microcode_field;
  program_counter := program_counter + 1;
end;

Procedure generate_ALU_operation(F,R,S:operand_type;opcode:longint);
var branch_field : array[1..4] of longint;
begin
  { store the branch field }
  branch_field[1] := am2910_opcode;
  branch_field[2] := branch_opcode;
  branch_field[3] := branch_address;
  branch_field[4] := branch_lookahead_buffer[2];
  am2910_opcode := cont;
  branch_opcode := 0;
  branch_address := 0;
  branch_lookahead_buffer[2] := no_branch;
  check_index_register(F,R,S);
  check_pipeline_stage(F,R,S,opcode);
  { restore the branch field }
  am2910_opcode := branch_field[1];
  branch_opcode := branch_field[2];
  branch_address := branch_field[3];
  branch_lookahead_buffer[2] := branch_field[4];
  display_ALU_operation(F,R,S,opcode);
  bind_ALU_opcode(opcode);
  bind_AF_AR_AS(F,R,S);
  bind_AIF_AIR_AIS(F,R,S);
  microcode_address := program_counter;
  output_microcode_field;
  program_counter := program_counter+1;
end; { of generate_ALU_operation }

Procedure check_F_bus(operand : operand_type);
begin
  if (write_lookahead_buffer[0].id <> blank_token) then
    if (operand.id = write_lookahead_buffer[0].id) then
      if (operand.index = write_lookahead_buffer[0].index) then
        if (operand.offset = write_lookahead_buffer[0].offset) then
```

```
        generate_nop;
  if (write_lookahead_buffer[0].id <> blank_token) then
    if (operand.id = write_lookahead_buffer[0].id) then
      if (operand.index = write_lookahead_buffer[0].index) then
        if (operand.offset = write_lookahead_buffer[0].offset) then
          generate_nop;
end; { of check_F_bus }

Procedure Clear_pipeline_stage;
begin
  while (write_lookahead_buffer[0].id <> blank_token) or
        (write_lookahead_buffer[1].id <> blank_token)  do
    generate_Nop;
  if (AIR[0] <> 7fffH) then generate_nop;
  if (AIS[0] <> 7fffH) then generate_nop;
  if (AIF[0] <> 7fffH) then generate_nop;
end; { of clear_pipeline_stage }

Procedure load_index_register(index : token_type);
var integer_index : token_type;
    t1,t2 : operand_type;
    index_pointer : longint;
begin
  { find an empty index register }
  index_pointer := 0;
  while (index_pointer <= max_index_register) and
          (index_register[index_pointer] <> blank_token) do
  begin
    index_pointer := index_pointer+1;
  end;
  { if no index register is available, flag as error }
  if index_pointer > max_index_register then
    write_error('index register is full  -                          ',blank_token);
  index_register[index_pointer] := index;
  if (index <> blank_token) and (index[1] <> '0') then
  begin
    assign_temp_variable(integer_index);
    reset_operand(t2);
    reset_operand(t1);
    t2.id := integer_index;
    t1.id := index;
    generate_ALU_operation(t2,t1,zero_operand,unary_round);
    clear_pipeline_stage;
    delete(integer_index,1,1);
    val_integer(integer_index,AS,i);
    AR := AS;
    write(outfile,';',program_counter,' load index_register['
                                                ,index_pointer,'] = ');
    write_token (outfile, index);
    write (outfile,'(');
    write_token(outfile, integer_index);
```

```
    writeln(outfile,')');
    AIF[1] := index_pointer;
    AIS[1] := index_pointer;
    AIR[1] := index_pointer;
    IA2[2] := 0;
    IA1 := 0;
    IA0 := 0;
    read_opcode := 1;
    microcode_address := program_counter;
    output_microcode_field;
    program_counter := program_counter+1;
  end;
end; { of load_index_register }

Procedure check_index_register(var F,R,S : operand_type);
{ 1. check existing index register
  2. if index not available, load required indices into the index register }
begin
  R.index_address := assign_index(R.index);
  S.index_address := assign_index(S.index);
  F.index_address := assign_index(F.index);
end; { check_index_register }

Procedure check_pipeline_stage(var F,R,S:operand_type;var opcode:longint);
begin
  { check to see if AIR[0] and AIS[0] are available for indexing }
  if (AIR[0] <> 7fffH) or (AIS[0] <> 7fffH) then generate_nop;
  { Note that F(k-2) is equivalent to write_lookahead_buffer[0].
    Since F(k-2) is not available on the R-port, we can swap R and S operands
    with proper conditions: }
  if (opcode = unary_minus) then
  begin
    swap_operand(R,S);    { swap R(k) and S(k) }
    opcode := subtraction;
  end; { of swap operand if operation is r_minus or unary_minus }

  { check to see if feedback paths can be used for calculation }

  { Case 1 }
  { R(k) = F(k-1) }
  if (R.id = write_lookahead_buffer[1].id) and
     (R.index = write_lookahead_buffer[1].index) and
     (R.offset = write_lookahead_buffer[1].offset) then
  begin
    { Case 1.1 }
    { R(k) = S(k) }
    if (R.id = S.id) and
       (R.index = S.index) and
       (R.offset = S.offset) then
    begin
      I4 := 1;
```

```
  I3[1] := 1;
  S.id := blank_token;
  S.id[1] := '~';
  S.id[2] := 'S';
  R.id := blank_token;
  R.id[1] := '~';
  R.id[2] := 'R'
end
else
{ Case 1.2 }
{ S(k) = F(k-2) }
if (S.id = write_lookahead_buffer[0].id) and
   (S.index = write_lookahead_buffer[0].index) and
   (S.offset = write_lookahead_buffer[0].offset) then
begin
   { Case 1.2.1 }
   { R(k) is a temporary variable }
  if (R.id[1] = '#') then
  begin
  S.id := blank_token;
  S.id[1] := '~';
  S.id[2] := 'F';
  R.id := blank_token;
  R.id[1] := '~';
  R.id[2] := 'R';
    ENF_bar := 1;
    I4 := 1;
    I3[1] := 1;
  end { of 1.2.1 }
  else
  { Case 1.2.2 }
  { R(k) is a permanent variable }
  begin
    generate_nop; { make R(k) <-- F(k-2) }
    { Case 1.2.2.1 }
    { R(k) and S(k) can be swapped }
    if (opcode = multiplication) or
       (opcode = addition) then
    begin
      swap_operand(R,S);  { swap R(k) and S(k) }
      S.id := blank_token;
      S.id[1] := '~';
      S.id[2] := 'F';
      ENF_bar := 1;
      I3[1] := 1;
    end { of 1.2.2.1 }
    else
    { Case 1.2.2.2 }
    { R(k) and S(k) cannot be swapped }
    begin
      generate_NOP; { make R and S available from memory }
```

```
      end; { of 1.2.2.2 }
    end { of 1.2.2 R(k) is a permanent variable }
  end { of 1.2  S(k) = F(k-2) }
  else
  { Case 1.3 }
  { S(k) <> F(k-1) and S(k) <> F(k-2) }
  begin
    R.id := blank_token;
    R.id[1] := '~';
    R.id[2] := 'R';
    I4 := 1;
  end; { of (1.3) R(k-1) = F(k-1) and S(k) <> F(k-2) }
end { of (1) R(k = F(k-1) }
else
{ Case 2 }
{ S(k) = F(k-1) }
if (S.id = write_lookahead_buffer[1].id) and    { S(k) = F(k-1) }
   (S.index = write_lookahead_buffer[1].index) and
   (S.offset = write_lookahead_buffer[1].offset) then
begin
  { Case 2.1 }
  { R(k) = F(k-2) }
  if (R.id = write_lookahead_buffer[0].id) and
     (R.index = write_lookahead_buffer[0].index) and
     (R.offset = write_lookahead_buffer[0].offset) then
  begin
    { Case 2.1.1 }
    { R(k) and S(k) can be swapped }
    if (opcode = multiplication) or
       (opcode = addition) then
    begin
      { Case 2.1.1.1 }
      { S(k) is a temporary variable }
      if S.id[1] = '#' then
      begin
        swap_operand(R,S);     { swap  R and S }
        S.id := blank_token;
        S.id[1] := '~';
        S.id[2] := 'F';
        ENF_bar := 1;
        I3[1] := 1;
        R.id := blank_token;
        R.id[1] := '~';
        R.id[2] := 'R';
        I4 := 1;
      end { of 2.1.1.1 }
      else
      { Case 2.1.1.2 }
      { S(k) is a permanent variable }
      begin
        generate_nop; { make R(k) available from memory and S(k) = F(k-2) }
```

```
          S.id := blank_token;
          S.id[1] := '-';
          S.id[2] := 'F';
          ENF_bar := 1;
          I3[1] := 1;
        end { of 2.1.1.2 }
      end { of 2.1.1 R(k) and S(k) can be swapped }
      else
      { Case 2.1.2 }
      { R(k) and S(k) cannot be swapped }
      begin
        generate_Nop; { make S(k) = F(k-2) }
          S.id := blank_token;
          S.id[1] := '-';
          S.id[2] := 'F';
        ENF_bar := 1;
        I3[1] := 1;
      end { of 2.1.2 }
    end { of 2.1 R(k) = F(k-2) }
    else
    { Case 2.2 }
    { R(k) <> F(k-2 }
    begin
      I3[1] := 1;
      S.id := blank_token;
      S.id[1] := '-';
      S.id[2] := 'S';
    end { of 2.2 }
  end { of 2 S(k) = F(k-1) }
  else
  { R(k) = F(k-2) }
  if (R.id = write_lookahead_buffer[0].id) and
     (R.index = write_lookahead_buffer[0].index) and
     (R.offset = write_lookahead_buffer[0].offset) then
    generate_nop  { make R(k) available from memory }
  else
  { S(k) = F(k-2) }
  if (S.id = write_lookahead_buffer[0].id) and
     (S.index = write_lookahead_buffer[0].index) and
     (S.offset = write_lookahead_buffer[0].offset) then
    generate_nop { make S(k) available from memory }

end; { of ckeck_pipeline_stage }

procedure display_ALU_operation(F,R,S : operand_type;opcode : longint);
begin
  write(outfile,';',program_counter,': ');
  write_token(outfile,F.id);
  if F.index <> blank_token then
  begin
    write(outfile,'[');
```

```
    write_token(outfile,F.index);
    if F.offset > 0 then
      write(outfile,'+',F.offset,']')
    else
    if F.offset < 0 then
      write(outfile,F.offset,']')
    else
      write(outfile,']');
  end
  else
  if F.offset <> 0 then
    write(outfile,'[',F.offset,']');
  write(outfile,' := ');
  write_token(outfile,R.id);
  if R.index <> blank_token then
  begin
    write(outfile,'[');
    write_token(outfile,R.index);
    if R.offset > 0 then
      write(outfile,'+',R.offset,']')
    else
    if R.offset < 0 then
      write(outfile,R.offset,']')
    else
      write(outfile,']');
  end
  else
  if R.offset <> 0 then
    write(outfile,'[',R.offset,']');
  case opcode of
    addition       : begin
                       write(outfile,' + ');
                     end;
    subtraction    : begin
                       write(outfile,' - ');
                     end;
    multiplication : begin
                       write(outfile,' * ');
                     end;
    unary_float    : begin
                       write(outfile,' float ');
                     end;
    unary_round    : begin
                       write(outfile,' round ');
                     end;
    unary_trunc    : begin
                       write_error(' truncation is not supported
                                                          ,blank_token);
                     end;
    r_minus :        begin
                       write(outfile,' - ');
```

```
                  end;
    or_operation:     begin
                        write(outfile,' or(+) ');
                      end;
    and_operation:    begin
                        write(outfile,' and(*) ');
                      end;
    otherwise         begin
                        writeln (errorfile);
                        write(errorfile,'unsupported real operator : ', opcode);
                        error_found
                      end
 end; { of case opcode }
  write_token(outfile,S.id);
  if S.index <> blank_token then
  begin
    write(outfile,'[');
    write_token(outfile,S.index);
    if S.offset > 0 then
      write(outfile,'+',S.offset,']')
    else
    if S.offset < 0 then
      write(outfile,S.offset,']')
    else
      write(outfile,']');
  end
  else
  if S.offset <> 0 then
    write(outfile,'[',S.offset,']');
  writeln(outfile);
end;  { of display_ALU_operation }

procedure bind_ALU_opcode(opcode:longint);
begin
  case opcode of
    addition       : mc325_buffer[1] := 0;
    subtraction    : mc325_buffer[1] := 1;

    multiplication : mc325_buffer[1] := 2;
    unary_float    : mc325_buffer[1] := 4;
    unary_round    : mc325_buffer[1] := 5;
    unary_trunc    : write_error(' truncation is not supported
                                                        ,blank_token);
    r_minus        : mc325_buffer[1] := 1;
    or_operation   : mc325_buffer[1] := 0;
    and_operation  : mc325_buffer[1] := 0;
    otherwise begin
                writeln(errorfile);
                writeln(errorfile,'unsupported real operator : ',opcode);
                error_found;
              end;
```

```
  end; { of case opcode }
end; { of bind_ALU_opcode }

procedure bind_AIF_AIR_AIS(F,R,S : operand_type);
{ bind the instruction fields AIF, AIR, AIS }
begin
  if (F.index <> blank_token) and (F.index[1] <> '0') then
  begin
    AIF[2] := F.index_address;
    IA2[2] := 1;
  end;
  if (R.index <> blank_token) and (R.index[1] <> '0') then
  begin
    AIR[0] := R.index_address;
    IA1 := 1;
  end;
  if (S.index <> blank_token) and (S.index[1] <> '0') then
  begin
    AIS[0] := S.index_address;
    IA0 := 1;
  end;
end; { of bind_AIF_AIR_AIS }

Procedure bind_AF_AR_AS(F,R,S:operand_type);
begin
  write_lookahead_buffer[2] := F;
  temp_token := blank_token;
  add_char_to_string (temp_token, '|');
  if (F.id[1] = '#') or (F.id[1] = '&') then
  begin
    delete(F.id,1,1);
    val_integer(F.id,AF[2],i);
  end
  else
    if F.id = temp_token then
      AF[2] := stack_pointer
    else
      if F.id[1] <> '~' then
      begin
        find_symbol(F.id,symbol_type,symbol_value,found);
        if not found then
          write_error('unknown id :                                        ',F.id);
        if symbol_value < 0 then
        begin
          { parameter passing call by value }
          AIF[2] := abs(symbol_value) - 1;
          IA2[2] := 1;
          AF[2] := 0;
        end
        else
          AF[2] := symbol_value+F.offset
```

```
    end;
temp_token := blank_token;
add_char_to_string (temp_token, '|');
if (R.id[1] = '#') or (R.id[1] = '&') then
begin
  delete(R.id,1,1);
  val_integer(R.id,AR,i);
end
else
  if R.id = temp_token then
    AR := stack_pointer
else
  if R.id[1] <> '-' then
  begin
    find_symbol(R.id,symbol_type,symbol_value,found);
    if not found then
       write_error('unknown id :                                   ',R.id);
    if symbol_value < 0 then
    begin
      { parameter passing call by value }
      AIR[0] := abs(symbol_value)-1;
      IA1 := 1;
      AR := 0;
    end
    else
      AR := symbol_value+R.offset;
  end;
temp_token := blank_token;
add_char_to_string (temp_token, '|');
if (S.id[1] = '#') or (S.id[1] = '&') then
begin
  delete(S.id,1,1);
  val_integer(S.id,AS,i);
end
else
  if S.id = temp_token then
    AS := stack_pointer
else
if S.id[1] <> '-' then
begin
  find_symbol(S.id,symbol_type,symbol_value,found);
  if not found then
     write_error('unknown id:                                   ',S.id);
  if symbol_value < 0 then
  begin
    { parameter passing call by value }
    AIS[0] := abs(symbol_value) - 1;
    IA0 := 1;
    AS := 0;
  end
  else
```

```
    AS := symbol_value+S.offset;
  end;
end; { of bind_AF_AR_and_AS }

{ Procedure load_sequencer_counter(token:token_type);
begin
  if token <> '&' then
    find_symbol(token,symbol_type,symbol_value,found);
  if (symbol_type <> integer_symbol_type) and
     (symbol_type <> integer_constant_symbol_type) and
     (token <> '&') then
  begin
    write_error('simple integer is expected.                    ',token);
  end;
  if token = integer_lookahead_buffer[1] then generate_nop;
  if token = integer_lookahead_buffer[0] then generate_nop;
  writeln(outfile,'; load sequencer counter : ',token);
  Dsel := 1;
  AS := symbol_value;
  AR := AS;
  output_microcode_field;
end;  }


Function search_branch_address_pointer(expression : expression_pointer):longint;
begin
  i := 1;
  while (expression^.address[i] <> 7fffH) do
  begin
    i := i+1;
    if i > max_branch_pointer then
    begin
      write_error('maximum branch pointer exceeded                  ',token);
    end;
  end;
  search_branch_address_pointer := i;
end; { of search_branch_address_pointer }

Procedure find_branch_address(var expression : expression_pointer;
                                                  var branch_state : longint);

var current_expression : expression_pointer;
    address_found,hold_position : boolean;
    negation_state : longint; { even : no negation
                                odd  : require negation }
    change_branch_state : longint; { even : no change of branch state
                                     odd  : require change of branch state }
   { branch_state --->      1 = branch if condition is true
                            2 = branch if condition is false
                            3 = true and false address jump may be required
                                ( last boolean expression evaluated )
```

```
                           4 = branch if condition is true
                           5 = branch if condition is false }
begin
  branch_state := 0;
  negation_state := 0;
  change_branch_state := 0;
  current_expression := expression;
  address_found := false;
  { find the branch state }
  repeat
    if (current_expression^.right <> nil) then
    begin
     if current_expression^.right^.operator = or_operation then
       branch_state := 1
     else
     if current_expression^.right^.operator = and_operation then
       branch_state := 2
     else
     if current_expression^.right^.operator = unary_not then
       change_branch_state := change_branch_state + 1
     else
     begin
       write_error('error found in generating short circuit jump      ',token);
     end;
    end;
    if branch_state = 0 then
    begin
      if current_expression^.up <> nil then
        current_expression := current_expression^.up
      else
      if current_expression^.left <> nil then
      begin
        current_expression := current_expression^.left;
        if (current_expression^.id[1]='#') or
           (current_expression^.id[1]='%') or
           (current_expression^.id[1]='&') then
          current_expression := current_expression^.up;
      end
      else
         branch_state := 3;
    end;
  until (branch_state > 0);
  { the top of the tree has reached if branch_state=3 }
  if (branch_state=3) then
  begin
    i := search_branch_address_pointer(current_expression);
    if odd(change_branch_state) then
    begin
      branch_state := 2;
      current_expression^.address[i] := (program_counter-1);
    end
```

```
    else
    begin
      branch_state := 2;
      current_expression^.address[i] := -(program_counter-1);
    end;
    current_expression^.address[i+1] := 7fffH;
{   writeln(outfile,'First gate : ------');
    writeln(outfile,current_expression^.id); }
  end
  else
  { find the branch address }
  begin
    negation_state := branch_state;
    repeat
      hold_position := false;
      if current_expression^.up <> nil then
        current_expression := current_expression^.up
      else
      if current_expression^.left <> nil then
      begin
        current_expression := current_expression^.left;
        if (current_expression^.id[1]='#') or
           (current_expression^.id[1]='%') or
           (current_expression^.id[1]='&') then
          current_expression := current_expression^.up
        else
          hold_position := true;
      end
      else
        branch_state := branch_state+3;

      if current_expression^.right <> nil then
      begin
        if (odd(negation_state) and (current_expression^.right^.operator=and_operation)) or
           (not odd(negation_state) and (current_expression^.right^.operator=or_operation)) then
        begin
          current_expression := current_expression^.right;
          if not hold_position then
          begin
            while (current_expression^.down <> nil) do
            begin
              current_expression := current_expression^.down;
              while (current_expression^.down <> nil) do
                current_expression := current_expression^.down;
              if (current_expression^.right <> nil) then
                current_expression := current_expression^.right;
            end;
          end;
          i := search_branch_address_pointer(current_expression);
          current_expression^.address[i+1] := 7fffH;
          current_expression^.address[i] := program_counter-1;
```

```
          address_found := true;
{          writeln(outfile,'Second gate : ------');
          writeln(outfile,current_expression^.id); }
        end
        else
        if (current_expression^.right^.operator = unary_not) then
          negation_state := negation_state+1;
      end;
    until (address_found) or (branch_state>2);
    if (branch_state>2) then
    begin
      branch_state := branch_state-3;
      negation_state := negation_state + 1;
      i := search_branch_address_pointer(current_expression);
      if odd(negation_state) then
          current_expression^.address[i] := -(program_counter-1)
      else
          current_expression^.address[i] := program_counter-1;
      current_expression^.address[i+1] := 7fffH;
{     writeln(outfile,'Third gate : ------');
      writeln(outfile,current_expression^.id); }
    end;
    if odd(change_branch_state) then
    begin
      if branch_state = 1 then branch_state := 2
                          else branch_state := 1;
    end;
  end;
end; { of find_branch_address }

procedure generate_boolean_assignment_microcode(var expression :
                                                    expression_pointer;
                                                    branch_true : boolean);
var F,R,S : operand_type;
begin
  reset_operand(R);
  reset_operand(S);
  if (branch_true) then str_real(0.0,R.id) else str_real(1.0,R.id);
  str_real(0.0,S.id);
  am2910_opcode :=  CJP;
  branch_lookahead_buffer[0] := unconditional;
  branch_address := program_counter+2;
  F.id := expression^.left^.up^.id;
  F.index := expression^.left^.up^.index;
  F.offset := expression^.left^.up^.offset;
  generate_ALU_operation(F,R,S,addition);
  if (branch_true) then str_real(1.0,R.id) else str_real(0.0,R.id);
  generate_ALU_operation(F,R,S,addition);
end; { of generate_boolean_assignment_microcode }

Procedure generate_branch_address(var expression : expression_pointer);
```

```
var branch_true : boolean;
    i : integer;
begin
  if expression^.address[1] <> 7fffH then
  begin
    while (branch_lookahead_buffer[0] <> 7fffH) or
          (branch_lookahead_buffer[1] <> 7fffH) do
      generate_Nop;
    i := 1;
    repeat
      writeln(outfile,'b ',abs(expression^.address[i])+2,' ',program_counter);
      expression^.address[i] := 7fffH;
      i := i+1;
    until expression^.address[i] = 7fffH;
  end;
  if (expression^.left^.up^.up=nil) and
     (expression^.left^.up^.left=nil) then
  begin
    if (expression^.left^.up^.id[1] <> '#') and
       (expression^.left^.up^.address[1] <> 7fffH) and
       ( (((expression^.left^.id[1]='#') or
           (expression^.left^.id=blank_token)) and
          ((expression^.id[1]='#') or (expression^.id=blank_token))) or
          ((expression^.left^.id[1]<>'#') and (expression^.id[1]<>'#')) ) then
    begin
      while (branch_lookahead_buffer[0] <> 7fffH) or
            (branch_lookahead_buffer[1] <> 7fffH) do
        generate_Nop;
      i := search_branch_address_pointer(expression^.left^.up);
      if (expression^.left^.up^.address[i-1]>0) then branch_true := true
                                               else branch_true := false;
      i := 1;
      repeat
        if (branch_true) then
        begin
          if expression^.left^.up^.address[i] > 0 then
            writeln(outfile,'b ',expression^.left^.up^.address[i]+2,' ',program_counter+1)
          else
            writeln(outfile,'b ',abs(expression^.left^.up^.address[i])+2,' ',program_counter);
        end
        else
        begin
          if expression^.left^.up^.address[i] > 0 then
            writeln(outfile,'b ',expression^.left^.up^.address[i]+2,' ',program_counter)
          else
            writeln(outfile,'b ',abs(expression^.left^.up^.address[i])+2,' ',program_counter+1);
        end;
        expression^.left^.up^.address[i] := 7fffH;
        i := i+1;
      until (expression^.left^.up^.address[i] = 7fffH);
      if (branch_true) then
```

```
        generate_boolean_assignment_microcode(expression,true)
      else
        generate_boolean_assignment_microcode(expression,false);
    end;
  end;
end; { of generate_branch_address }

procedure swap_operand(var R,S:operand_type);
var temp : operand_type;
begin
  temp := R;
  R := S;
  S := temp;
end; { of procedure swap_operand }

procedure assign_temp_boolean_variable(var F : operand_type);
begin
  str_integer(dataram_address_limit-temp_variable_limit,F.id);
  temp_token := blank_token;
  temp_token[1] := '#';
  concat (temp_token, F.id);
  F.id := temp_token
end; { of assign_temp_boolean_variable }

procedure generate_greater_than(var expression : expression_pointer);
var F,R,S : operand_type;
    opcode,branch_state : longint;
begin
  fetch_expression(F,R,S,expression);
  swap_operand(R,S);
  opcode := expression^.operator;
  check_pipeline_stage(F,R,S,opcode);
  generate_branch_address(expression);
  assign_temp_boolean_variable(F);
  generate_ALU_operation(F,R,S,subtraction);
  find_branch_address(expression^.left^.up,branch_state);
  if branch_state = 1 then
    branch_lookahead_buffer[1] := if_negative
  else
    branch_lookahead_buffer[1] := if_not_negative;
  generate_branch_address(expression);
end; { of generate_greater_than }

procedure generate_less_than(var expression : expression_pointer);
var F,R,S : operand_type;
    opcode,branch_state : longint;
begin
  fetch_expression(F,R,S,expression);
  opcode := expression^.operator;
  check_pipeline_stage(F,R,S,opcode);
  generate_branch_address(expression);
```

```
  assign_temp_boolean_variable(F);
  generate_ALU_operation(F,R,S,subtraction);
  find_branch_address(expression^.left^.up,branch_state);
  if branch_state = 1 then
    branch_lookahead_buffer[1] := if_negative
  else
    branch_lookahead_buffer[1] := if_not_negative;
  generate_branch_address(expression);
end; { of generate_less_than }

procedure generate_equal(var expression : expression_pointer);
var F,R,S : operand_type;
    opcode,branch_state : longint;
begin
  fetch_expression(F,R,S,expression);
  opcode := expression^.operator;
  check_pipeline_stage(F,R,S,opcode);
  generate_branch_address(expression);
  assign_temp_boolean_variable(F);
  generate_ALU_operation(F,R,S,subtraction);
  find_branch_address(expression^.left^.up,branch_state);
  if branch_state = 1 then
    branch_lookahead_buffer[1] := if_zero
  else
    branch_lookahead_buffer[1] := if_not_zero;
  generate_branch_address(expression);
end; { of generate_equal }

procedure generate_not_equal(var expression : expression_pointer);
var F,R,S : operand_type;
    opcode,branch_state : longint;
begin
  fetch_expression(F,R,S,expression);
  opcode := expression^.operator;
  check_pipeline_stage(F,R,S,opcode);
  generate_branch_address(expression);
  assign_temp_boolean_variable(F);
  generate_ALU_operation(F,R,S,subtraction);
  find_branch_address(expression^.left^.up,branch_state);
  if branch_state = 1 then
    branch_lookahead_buffer[1] := if_not_zero
  else
    branch_lookahead_buffer[1] := if_zero;
  generate_branch_address(expression);
end; { of generate_not_equal }

procedure generate_greater_than_or_equal(var expression : expression_pointer);
var F,R,S : operand_type;
    opcode,branch_state : longint;
begin
  fetch_expression(F,R,S,expression);
```

```
  opcode := expression^.operator;
  check_pipeline_stage(F,R,S,opcode);
  generate_branch_address(expression);
  assign_temp_boolean_variable(F);
  generate_ALU_operation(F,R,S,subtraction);
  find_branch_address(expression^.left^.up,branch_state);
  if branch_state = 1 then
    branch_lookahead_buffer[1] := if_not_negative
  else
    branch_lookahead_buffer[1] := if_negative;
  generate_branch_address(expression);
end; { of generate_greater_than_or_equal }

procedure generate_less_than_or_equal(var expression : expression_pointer);
var F,R,S : operand_type;
    opcode,branch_state : longint;
begin
  fetch_expression(F,R,S,expression);
  swap_operand(R,S);
  opcode := expression^.operator;
  check_pipeline_stage(F,R,S,opcode);
  generate_branch_address(expression);
  assign_temp_boolean_variable(F);
  generate_ALU_operation(F,R,S,subtraction);
  find_branch_address(expression^.left^.up,branch_state);
  if branch_state = 1 then
    branch_lookahead_buffer[1] := if_not_negative
  else
    branch_lookahead_buffer[1] := if_negative;
  generate_branch_address(expression);
end; { of generate_less_than_or_equal }

Procedure generate_unary_not(var expression : expression_pointer);
var F,R,S : operand_type;
    opcode,branch_state : longint;
begin
  fetch_expression(F,R,S,expression);
  opcode := expression^.operator;
  if R.id[1] = '#' then
  begin
    generate_branch_address(expression);
  end
  else
  begin
    if f.id[1] = '#' then
    begin
      reset_operand(S);
      str_real(0.0,S.id);
      check_pipeline_stage(F,R,S,opcode);
      generate_branch_address(expression);
      assign_temp_boolean_variable(F);
```

```
      generate_ALU_operation(F,R,S,addition);
      find_branch_address(expression^.left^.up,branch_state);
      if branch_state = 1 then
        branch_lookahead_buffer[1] := if_zero
      else
        branch_lookahead_buffer[1] := if_not_zero;
      generate_branch_address(expression);
    end
    else
    begin
      S := R;
      reset_operand(R);
      str_real(1.0,R.id);
      generate_ALU_operation(F,R,S,subtraction);
    end;
  end;
end; { of generate_unary_not }

procedure generate_and(var expression : expression_pointer);
var F,R,S : operand_type;
    branch_state,opcode : longint;
begin
  fetch_expression(F,R,S,expression);
  if ((S.id[1] <> '#') and
      (R.id[1] <> '#')) then
  begin
    if (F.id[1]='#') then
    begin
      opcode := multiplication;
      check_pipeline_stage(F,R,S,opcode);
      generate_branch_address(expression);
      assign_temp_boolean_variable(F);
      generate_ALU_operation(F,R,S,opcode);
      find_branch_address(expression^.left^.up,branch_state);
      if branch_state = 1 then
        branch_lookahead_buffer[1] := if_not_zero
      else
        branch_lookahead_buffer[1] := if_zero;
    end
    else
      generate_ALU_operation(F,R,S,multiplication);
  end
  else
  if (R.id[1] = '#') and (S.id[1] = '#') then
  begin
    generate_branch_address(expression);
  end
  else
  if (R.id[1] = '#') then
  begin
    opcode := addition;
```

```
    str_real(0.0,R.id);
    check_pipeline_stage(F,R,S,opcode);
    generate_branch_address(expression);
    assign_temp_boolean_variable(F);
    generate_ALU_operation(F,R,S,opcode);
    find_branch_address(expression^.left^.up,branch_state);
    if branch_state = 1 then
      branch_lookahead_buffer[1] := if_not_zero
    else
      branch_lookahead_buffer[1] := if_zero;
    expression^.id[1] := '#';
    generate_branch_address(expression);
  end
  else
  if (S.id[1] = '#') then
  begin
    opcode := addition;
    str_real(0.0,S.id);
    check_pipeline_stage(F,R,S,opcode);
    generate_branch_address(expression);
    assign_temp_boolean_variable(F);
    generate_ALU_operation(F,R,S,opcode);
    find_branch_address(expression^.left^.up,branch_state);
    if branch_state = 1 then
      branch_lookahead_buffer[1] := if_not_zero
    else
      branch_lookahead_buffer[1] := if_zero;
    expression^.left^.id[1] := '#';
    generate_branch_address(expression);
  end;
end; { of generate_and }

procedure generate_or(var expression : expression_pointer);
var F,R,S : operand_type;
    branch_state,opcode : longint;
begin
  fetch_expression(F,R,S,expression);
  opcode := addition;
  if ((S.id[1] <> '#') and
      (R.id[1] <> '#')) then
  begin
    check_pipeline_stage(F,R,S,opcode);
    generate_branch_address(expression);
    assign_temp_boolean_variable(F);
    generate_ALU_operation(F,R,S,opcode);
    find_branch_address(expression^.left^.up,branch_state);
    if branch_state = 1 then
      branch_lookahead_buffer[1] := if_not_zero
    else
      branch_lookahead_buffer[1] := if_zero;
    expression^.id[1] := '#';
```

```
    expression^.left^.id[1] := '#';
    generate_branch_address(expression);
  end
  else
  if (R.id[1] = '#') and (S.id[1] = '#') then
  begin
    generate_branch_address(expression);
  end
  else
  if (R.id[1] = '#') then
  begin
    str_real(0.0,R.id);
    check_pipeline_stage(F,R,S,opcode);
    generate_branch_address(expression);
    assign_temp_boolean_variable(F);
    generate_ALU_operation(F,R,S,opcode);
    find_branch_address(expression^.left^.up,branch_state);
    if branch_state = 1 then
      branch_lookahead_buffer[1] := if_not_zero
    else
      branch_lookahead_buffer[1] := if_zero;
    expression^.id[1] := '#';
    generate_branch_address(expression);
  end
  else
  if (S.id[1] = '#') then
  begin
    str_real(0.0,S.id);
    check_pipeline_stage(F,R,S,opcode);
    generate_branch_address(expression);
    assign_temp_boolean_variable(F);
    generate_ALU_operation(F,R,S,opcode);
    find_branch_address(expression^.left^.up,branch_state);
    if branch_state = 1 then
      branch_lookahead_buffer[1] := if_not_zero
    else
      branch_lookahead_buffer[1] := if_zero;
    expression^.left^.id[1] := '#';
    generate_branch_address(expression);
  end;
end; { of generate_or }

function assign_index(index : token_type):longint;
var index_pointer : longint;
begin
  if index <> blank_token then
  begin
    index_pointer := 0;
    while (index_pointer <= max_index_register) and
          (index <> index_register[index_pointer]) do
    begin
```

```
      index_pointer := index_pointer+1;
    end;
    if index_pointer > max_index_register then
    begin
      load_index_register(index);
      index_pointer := assign_index(index);
    end;
    assign_index := index_pointer;
  end
  else
    assign_index := 0;
end; { of assign_index }

Procedure generate_read_function(function_number : longint;
                                          fx,x : operand_type);
begin
  find_symbol(fx.id,fx.id_type,fx.id_address,found);
  if not found then
     write_error('unknow id:                              ',x.id);
  fx.index_address := assign_index(fx.index);
  find_symbol(x.id,x.id_type,x.id_address,found);
  if not found then
     write_error('unknow id:                              ',x.id);
  x.index_address := assign_index(x.index);
  clear_pipeline_stage;
  temp_token := blank_token;
  operand_string(fx, temp_token);
  write (outfile,'; read_function(');
  write_token(outfile,temp_token);
  write (outfile, ',');
  temp_token := blank_token;
  operand_string(x, temp_token);
  write_token (outfile,temp_token);
  writeln (outfile, ')');
  microcode_address := program_counter;
  AR := x.id_address + x.offset;
  if (x.index <> blank_token) and (x.index[1] <> '0') then
  begin
    AIR[0] := x.index_address;
    IA1 := 1;
  end;
  AS := AR;
  IA0 := IA1;
  output_microcode_field;
  program_counter := program_counter + 1;
  microcode_address := program_counter;
  AF[0] := fx.id_address + fx.offset;
  if (fx.index <> blank_token) and (fx.index[1] <> '0') then
  begin
    AIF[0] := fx.index_address;
    IA2[0] := 1;
```

```
  end;
  write_opcode :=  read_function_opcode + function_number;
  output_microcode_field;
  program_counter := program_counter + 1;
end; { of generate_read_function }.
```

```
File: COMPILER.PAS
module Compile;

$include(global.def)
$include(hex_conv.def)
$include(ieee_cnv.def)
$include(utility.def)
$include(init.def)
$include(fetch_tk.def)
$include(symbol_t.def)
$include(code_gen.def)
$include(exprsion.def)
$include(exprtree.def)
$include(declare.def)
$include(io.def)
$include(arith.def)
$include(stdprocd.def)
$include(mainbody.def)
$include(emu_lib.def)

program Compile (input, output);
begin
  initialize;
  { program heading }
  fetch_token;
  if token <> program_heading then
  begin
    writeln(errorfile);
    writeln(errorfile,'!!!! syntax error : program heading expected');
    error_found;
  end
  else
    program_heading_block;
  fetch_token;
  program_main_block;
  if (token[1] <> '.') then
  begin
    writeln(errorfile);
    writeln(errorfile,'!!!! syntax error : "." is expected');
    error_found;
  end;
  if program_counter > program_counter_limit then
  begin
    writeln(errorfile);
    writeln(errorfile,'!!!! Program is too long. Limit is ',program_counter_limit,'.');
    writeln(errorfile,'     Current program size is ',program_counter,'.');
    error_found;
  end;
  clear_pipeline_stage;
  microcode_address := program_counter;
  writeln(outfile,';',program_counter,' :  goto ',program_counter);
```

```
  am2910_opcode := CJP;
  branch_opcode := unconditional;
  branch_address := program_counter;
  output_microcode_field;
  writeln;
end.
```

File: DECLARE.DEF

```
public declare;
  procedure type_declaration_block;
  procedure var_declaration_block;
  procedure const_declaration_block;
  Procedure assign_real_constant(var operand:operand_type;value : real);
```

File: DECLARE.PAS

```
module declare;

$include(declare.def)
$include(emu_lib.def)
$include(global.def)
$include(fetch_tk.def)
$include(utility.def)
$include(symbol_t.def)

private declare;
procedure type_declaration_block;
begin
  writeln(errorfile);
  writeln(errorfile,'!!!! error : general type is not supported yet.');
  error_found;
end;

procedure var_declaration_block;
var i,j : longint;
    var_type : longint;
begin
  i := 0;
  fetch_token; { variable name }
  repeat
    i := i+1;
    symbol_array[i] := token; { insert token to symbol array };
    fetch_token; { , }
    while token = comma do
    begin
      fetch_token; { variable_name };
      i := i+1;
      symbol_array[i] := token; { insert token to symbol array };
      fetch_token;
    end;
    verify_token(token,colon);
    fetch_token; { variable_type }
    if (token = real_token) then
    begin
      var_type := real_symbol_type;
    end
    else
    if (token = integer_token) then
    begin
      var_type := integer_symbol_type;
    end
    else
    if (token = boolean_token) then
    begin
      var_type := boolean_symbol_type;
    end
```

```
else
if (token = array_token) then
begin
  fetch_token; { [ }
  verify_token(token,open_square_bracket);
  fetch_token; { constant }
  temp_token := blank_token;
  temp_token[1] := '-';
  if token = temp_token then
  begin
    fetch_token;
    array_lower_range := -1;
  end
  else
    array_lower_range := 1;
  if symbol_type = integer_constant_symbol_type then
  begin
    array_lower_range := integer_constant_value*array_lower_range;
  end
  else
  begin
    find_symbol(token,symbol_type,symbol_value,found);
    if symbol_type = integer_constant_symbol_type then
      array_lower_range := integer_constant_value*array_lower_range
    else
    begin
      write_error('integer constant expected                        ',token);
    end;
    fetch_token; { . }
    verify_token(token,dot);
  end;
  fetch_token; { . }
  verify_token(token,dot);
  fetch_token; { constant }
  if token[1] = '-' then
  begin
    fetch_token;
    array_upper_range := -1;
  end
  else array_upper_range := 1;
  if symbol_type = integer_constant_symbol_type then
  begin
    array_upper_range := integer_constant_value*array_upper_range;
  end
  else
  begin
    find_symbol(token,symbol_type,symbol_value,found);
    if symbol_type = integer_constant_symbol_type then
      array_upper_range := integer_constant_value*array_upper_range
    else
    begin
```

```
        write_error('integer constant expected                    ',token);
      end;
    end;
    fetch_token; { ] }
    verify_token(token,close_square_bracket);
    fetch_token; { of }
    verify_token(token,of_token);
    fetch_token; { real }
    if token = real_token then
    begin
      var_type := real_array_symbol_type;
    end
    else
    if token = integer_token then
    begin
      var_type := integer_array_symbol_type;
    end
    else
    if token = boolean_token then
    begin
      var_type := boolean_array_symbol_type;
    end
    else
    begin
      write_error('Unsupported array type                    ',token);
    end;
  end
  else
  begin
    write_error('syntax error : unsupported type              ',token);
  end;
  fetch_token; { ; }
  verify_token(token,semicolon);
  for j := i downto 1 do
  begin
    insert_symbol(symbol_array[j],var_type,symbol_value);
    no_local_variable[procedure_level] := no_local_variable[procedure_level]+1;
    local_variable[procedure_level,no_local_variable[procedure_level]] := symbol_array[j];
{   writeln(outfile,'symbol_value : ',symbol_value);
    writeln(outfile,'lower range  : ',array_lower_range);
    writeln(outfile,'upper range  : ',array_upper_range);
    writeln(outfile,'dataram_address : ',dataram_address);  }
  end;
  i := 0;
  fetch_token;
 until ( token = begin_token ) or (token = const_declaration) or
       ( token = function_heading ) or
       ( token = procedure_heading);
end; { of var_declaration_block }

procedure const_declaration_block;
```

```
var constant_name : token_type;
    invert_constant_value : boolean;
begin
  fetch_token;
  repeat
    invert_constant_value := false;
    constant_name := token;
    fetch_token;
    verify_token(token,equal_token);
    fetch_token; { constant value }
    if token = true_token then
    begin
      str_real(1.0,token);
      real_constant_value := 1.0;
      symbol_type := boolean_constant_symbol_type;
    end
    else
    if token = false_token then
    begin
      str_real(0.0,token);
      real_constant_value := 0;
      symbol_type := boolean_constant_symbol_type;
    end
    else
    if token[1] = '-' then
    begin
      fetch_token;
      temp_token := blank_token;
      temp_token[1] := '-';
      concat(temp_token,token);
      token := temp_token;
      invert_constant_value := true;
    end
    else
    if token[1] = '+' then
    begin
      fetch_token;
    end
    else
    if (symbol_type <> real_constant_symbol_type) and
       (symbol_type <> integer_constant_symbol_type) then
    begin
      find_symbol(token,symbol_type,symbol_value,found);
    end;
    if (symbol_type <> real_constant_symbol_type) and
       (symbol_type <> integer_constant_symbol_type) then
    begin
      write_error('invalid or unsupported constant                   ',token);
    end;
    if invert_constant_value then
    begin
```

```
      real_constant_value := -real_constant_value;
      integer_constant_value := -integer_constant_value;
    end;
    insert_symbol(constant_name,symbol_type,symbol_value);
    no_local_variable[procedure_level] := no_local_variable[procedure_level]+1;
    local_variable[procedure_level,no_local_variable[procedure_level]] := constant_name;
    declare_constant(symbol_value,symbol_type,constant_name);
    fetch_token;
    verify_token(token,semicolon);
    fetch_token;
  until ( token = begin_token ) or (token = var_declaration) or
        ( token = procedure_heading) or ( token = function_heading ) or
        ( token = const_declaration );
end; { of const_declaration_block }

Procedure assign_real_constant(var operand:operand_type;value : real);
{ This procedure is used to declare a constant real number.
  It is used to simplify a declaration of constant with real number.
  It is primarily used to aid the development of standard function calls.
   }
begin
  reset_operand(operand);
  str_real(value,operand.id);
  operand.id_type := real_constant_symbol_type;
  real_constant_value := value;
  find_symbol(operand.id,operand.id_type,operand.id_address,found);
  if not found then
  begin
    insert_symbol(operand.id,operand.id_type,operand.id_address);
    declare_constant(operand.id_address,operand.id_type,operand.id);
  end;
end; { of assign_real_constant }.
```

```
File: EMU_LIB.DEF
public emu_lib;
   procedure clearscreen;
   procedure gotoxy (x,y :longint);
   procedure str_integer ( integer_number : longint; var token : token_type);
   procedure str_real ( x : real; var token : token_type);
   procedure val_real ( token : token_type; var real_number : real;
                                          var error_integer : longint);
   procedure val_integer ( token : token_type; var number :
                                longint;  var error_integer : longint);
   function length ( token : token_type ) : longint;
   procedure add_char_to_string ( var token : token_type; x : char);
   procedure concat (var token1 : token_type; token2 : token_type);
   procedure delete (var token : token_type; index, counter : longint);
   procedure read_token ( var token : token_type);
   procedure write_token (var x : text; token : token_type);
```

File: EMU_LIB.PAS

```
module emu_lib;

$include(emu_lib.def)
$include(global.def)
$include(utility.def)

private emu_lib;

procedure clearScreen;
begin
  write(chr(27),'[2J');
end;


function length ( token : token_type ) : longint;
var
   i : longint;
begin
   i := 1;
   while (token[i] <> ' ') and (i < max_token_length) do
      i := i + 1;
   if token[max_token_length] <> ' ' then
      length := max_token_length
   else
      length := i - 1
end; { length }

procedure delete ( var token : token_type; index, counter : longint);
var
   i,j,l,x : longint;
begin
 if (index < 1) or (counter < 0) then
    write_error ('Parameters to procedure delete out of range.      ',
                                                         blank_token)
 else
 if index <= max_token_length then
    begin
       j := 1;
       i := index;
       l := length(token);
       while (i <= max_token_length) and (i < (index + counter)) do
          begin
             for x := index to (l - j) do
                token[x] := token[x + 1];
             if (j = 1) and (i = index) then
                token[l] := ' '
             else
                j := j + 1;
             i := i + 1;
          end
```

```
    end
end; { delete }

procedure gotoXY(x,y : longint);
begin
  if x>80 then x := 80;    if x<1 then x := 1;
  if y>24 then y := 24;    if y<1 then y := 1;
  write(chr(27),'[',y,';',x,'H');
end;


{procedure pad_with_spaces ( var string : token_type; a,b : longint);

var
   x : longint;

begin
   for x := a to b do
      string[x] := ' '
end;  pad_with_spaces }


procedure add_char_to_string ( var token : token_type ; x : char );

var
   i :longint;

begin
   i := 1;
   while (token[i] <> ' ') and  (i <= max_token_length) do
      i := i + 1;
   if i > max_token_length then
      write_error ('token too long                                    ',
                                                                        token)
   else
      token[i] := x
end; { add_char_to_string }


procedure concat ( var token1 : token_type ; token2 : token_type);

{ Append as much of token2 onto token1 as possible }

var
   i, j : longint;

begin
   j := 1;
   i := length(token1) + 1;
   while (i <= max_token_length) and (token2[j] <> ' ') do
      begin
```

```
        token1[i] := token2[j];
        i := i + 1;
        j := j + 1
      end;
end; { concat }




{procedure str_integer ( x : real; var string : token_type);
var
   i, j, k : longint;
   integer_number : longint;
   temp_char : char;

begin
   string := blank_token;
   i := 1;
   j := 1;
   if x = 0.0 then
      begin
         k := 2;
         string[1] := '0'
      end
   else
      begin
         if x < 0.0 then
            begin
               string[i] := '-';
               i := i + 1
            end;
         j := i;
        integer_number := abs(ltrunc(x));

         while integer_number > 0 do
            begin
               string[i] := chr ((integer_number mod 10) + 30H);
              integer_number := integer_number div 10;
               i := i + 1
            end;
         k := i;
         i := i - 1;
         while j < i do
            begin
               Temp_Char := string[i];
               string[i] := string[j];
               string[j] := temp_char;
               j := j + 1;
               i := i - 1
            end
```

```
      end
end;  str_integer }

procedure str_integer ( integer_number : longint; var token : token_type);
var
   i, j, k : integer;
   temp_char : char;

begin
   token := blank_token;
   i := 1;
   j := 1;
   if integer_number = 0 then
      begin
         k := 2;
         token[1] := '0'
      end
   else
      begin
         if integer_number < 0 then
            begin
               token[i] := '-';
               i := i + 1
            end;
         j := i;
         integer_number := abs(integer_number);
         while integer_number > 0 do
            begin
               token[i] := chr ((integer_number mod 10) + 30H);
               integer_number := integer_number div 10;
               i := i + 1
            end;
         k := i;
         i := i - 1;
         while j < i do
            begin
               Temp_Char := token[i];
               token[i] := token[j];
               token[j] := temp_char;
               j := j + 1;
               i := i - 1
            end;
      end;
end; { str_integer }


procedure str_real ( x : real; var token : token_type );

var
   base, mantissa, fraction : real;
   i, j, exponent  : longint;
```

```
   temp_token : token_type;

function x_to_the_y  ( x : real ; y : longint ) : real;

var
  i : longint;
  total : real;

begin
   total := 1;
   for i := 1 to y do
      total := total * x;
   x_to_the_y := total
end; { x_to_the_y }

begin
   token := blank_token;
   if x = 0 then
      exponent := 0
   else
      begin
         exponent := ltrunc (ln (abs(x))/ln(10));
         if (exponent < 1) and (abs(x) < 1) then
            exponent := exponent -1
      end;
   if exponent < 1 then
      base := 10
   else
      base := 0.1;
   mantissa := x * (x_to_the_y (base, abs(exponent)));
   str_integer (ltrunc (mantissa), temp_token);
   i := 1;
   token[i] := temp_token[i];
   if temp_token[2] <> ' ' then
      begin
         i:= i + 1;
         token[i] := temp_token[i]
      end;
    i := i + 1;
    token[i] := '.';
    fraction := abs(mantissa);
    for j := 1 to 10 do
       begin
          fraction := fraction - ltrunc(fraction);
          fraction := fraction * 10;
          str_integer (ltrunc(fraction), temp_token);
          token[i + j] := temp_token[1];
       end;
    i := i + j + 1;
    token[i] := 'e';
    str_integer (exponent, temp_token);
```

```
   for j := 1 to 3 do
     token[i + j] := temp_token[j];
end; { str_real }


procedure val_integer(token : token_type; var number : longint;
                                          var error_integer : longint);
label 1;
var i : longint;
    real_number, power : real;
begin
  error_integer := 0;
  i := 0;
  {check for valid longint}
  for i := 1 to max_token_length do
  begin
    if (token[i] <> ' ') and (token[i] <> '+') and (token[i] <> '-') then
         if ((ord(token[i]) < 48) or (ord(token[i]) > 57)) then
            begin
              error_integer := 9999;  { for error checking - T.F. }
              goto 1
            end;
  end;
  i := max_token_length;
  while (i > 0) and (token[i] = ' ') do i := i-1;
  if (i = 0) then goto 1;              { if token = blank token }
  real_number := ord(token[i])-48;
  power := 1;
  i := i-1;

  while (i > 0) do
    begin
      if ((ord(token[i]) >= 48) and (ord(token[i]) <= 57)) then
      begin
        power := power * 10.0;
        real_number := real_number + (wrd(token[i])-48) * power;
      end;
      if token[i] = '-' then real_number := -real_number;
      i := i-1;
    end;
1: number := ltrunc(real_number);
end; { val_integer }


procedure val_real (token : token_type; var real_number : real;
                                          var error_integer : longint);
label 1;

var i,j      : longint;
    exponent : longint;
    x        : real;
```

```
    sign     : longint;
    exponent_token : token_type;

{   Discarding leading zero  }

Procedure Delete_leading_zero (var token : token_type);
var i,j : longint;
begin
    i := 1;
    while i <= max_token_length do
    begin
      if (token[i] = '0') then
      begin
         for j := i to max_token_length-1 do
         begin
             token[j] := token[j+1];
         end;
         token[max_token_length] := ' ';
      end
      else
         if (token[i] = '+') or (token[i] = '-') then
             i := i + 1
         else
             i := max_token_length + 1;
   end;
end;


{ This function converts a packed array of character that represents an integer
  to real number }

function char_integer_to_real(token : token_type): real;
label 1;
var i : longint;
    x,power : real;
begin
    x := 0;

    {check for valid integer}
    for i := 1 to max_token_length do
    begin
        if (token[i] <> ' ') and (token[i] <> '+') and (token[i] <> '-') then
            if ((ord(token[i]) < 48) or (ord(token[i]) > 57)) then
            begin
               error_integer := 9999;       { for error checking - T.F.}
               goto 1
            end;
    end;

    i := max_token_length;
    while (i > 0) and (token[i] = ' ') do i := i-1;
```

```
    if (i = 0) then goto 1;          { if token = blank token }

    x := ord(token[i])-48;
    power := 1;
    i := i-1;

    while (i > 0) do
    begin
      if ((ord(token[i]) >= 48) and (ord(token[i]) <= 57)) then
      begin
         power := power*10.0;
         x := x + (ord(token[i])-48)*power;
      end;
      if token[i] = '-' then x := -x;
      i := i-1;
    end;

1:  char_integer_to_real := x;
end;   { of char integer to real conversion }


{ beginning of the function token to real converter }

begin
    exponent := 0;
    error_integer := 0;  { if stays 0 then no error occurred - T.F. }
    delete_leading_zero (token);

{ Detecting whether digit left of decimal point is 0 }

  if (token[1] = '.' ) or ((token[2] = '.') and
                           not (token[1] in ['1'..'9'])) then

    { + or - sign could have preceded the decimal point }

    begin
       if token[1] = '.' then i := 2
       else i := 3;

       while token[i] = '0' do
       begin
         exponent := exponent-1;
         i := i + 1;
       end
    end;

{  Detecting decimal point }

    i := 1;
    while (i <= max_token_length) do
    begin
```

```
      if token[i] = '.' then
      begin
         for j := i to max_token_length-1 do
             token[j] := token[j+1];
         token[max_token_length] := ' ';
         i := max_token_length;
      end
      else
      if (token[i] = 'e') or (token[i] = 'E') then
         i := max_token_length
      else
      if  (ord(token[i]) >= 48) and (ord(token[i]) <= 57) then
                  exponent := exponent+1;

      i := i + 1;
   end;

{  check for exponential notation 'e' or 'E' }

   i := 1;
   while (i <= max_token_length) do
   begin
      if (token[i] = 'e') or (token[i] = 'E') then
      begin
         token[i] := ' ';
         for j := 1 to max_token_length do
         begin
             exponent_token[j] :=  ' ';
         end;

         for j := i+1 to max_token_length do
         begin
             exponent_token[j] := token[j];
             token[j] := ' ';
         end;
         x := char_integer_to_real(exponent_token);
         exponent := exponent + round(x);

         i := max_token_length + 1;
      end
      else
         i := i + 1;
   end;

   x := char_integer_to_real(token);
   while abs(x) >= 1 do x := x/10;

   if exponent > 0 then
      for i := 1 to exponent do x := x*10.0;
   if exponent < 0 then
      for i := -1 downto exponent do x := x/10.0;
```

```
1: real_number := x;
end;

procedure write_token (var x : text; token : token_type );

var
   i : longint;

begin
   i := 1;
   while (i <= max_token_length) do
      begin
         if i = max_token_length then
           begin
             if token[i] <> ' ' then write(x,token[i]);
             i := i+1
           end
         else
         if ((token[i] = ' ') and (token[i + 1] = ' ')) then
               i := max_token_length + 1
         else
            begin
               write(x, token[i]);
               i := i + 1
            end
      end
end; { write_token_string }

procedure read_token ( var token : token_type );
var
   i : longint;
begin
   i := 1;
   token := blank_token;
   while (not eoln) and (i < max_token_length) do
      begin
         read (token[i]);
         i := i + 1
      end;
   readln;
end; { read_token }.
```

File: EXPRSION.DEF

```
public exprsion;
  procedure insert_child_expression(operand : operand_type;operator : longint);
  procedure insert_sibling_expression(operand : operand_type;operator:longint);
  procedure reset_last_expression(head_operand : operand_type);
  procedure node_display(expression : expression_pointer);
  procedure create_expression(var expression : expression_pointer);
  procedure delete_expression(var expression : expression_pointer);
  procedure fetch_expression_operand(expression : expression_pointer;var operand : operand_type);
  function assignment_necessary(var expression : expression_pointer):boolean;
  function index_assignment_necessary(var expression : expression_pointer): boolean;
```

```
File: EXPRSION.PAS
module exprsion;
$include(exprsion.def)
$include(global.def)
$include(utility.def)
$include(emu_lib.def)

private exprsion;
procedure insert_child_expression(operand : operand_type;operator : longint);
begin
  { last_expression has to be not equal to nil }
  last_expression[expression_level]^.id := operand.id;
  last_expression[expression_level]^.id_type := operand.id_type;
  last_expression[expression_level]^.index := operand.index;
  last_expression[expression_level]^.offset := operand.offset;
  last_expression[expression_level]^.operator := operator;
  create_expression(new_expression);
  last_expression[expression_level]^.right := new_expression;
  new_expression^.left := last_expression[expression_level];
  create_expression(new_expression);
  last_expression[expression_level]^.down := new_expression;
  new_expression^.up := last_expression[expression_level];
  last_expression[expression_level] := new_expression;
end; { of insert_child_expression }

procedure insert_sibling_expression(operand : operand_type;operator:longint);
begin
  { last_expression has to be not equal to nil }
  last_expression[expression_level]^.id := operand.id;
  last_expression[expression_level]^.id_type := operand.id_type;
  last_expression[expression_level]^.index := operand.index;
  last_expression[expression_level]^.offset := operand.offset;
  last_expression[expression_level]^.operator := operator;
  create_expression(new_expression);
  last_expression[expression_level]^.right := new_expression;
  new_expression^.left := last_expression[expression_level];
  last_expression[expression_level] := new_expression;
end;  { of right insert expression }

procedure reset_last_expression(head_operand : operand_type);
begin
  last_expression[expression_level] := first_expression[expression_level];
  expression_operator[expression_level] := null_operator;
  last_expression[expression_level]^.id := head_operand.id;
  last_expression[expression_level]^.id_type := head_operand.id_type;
  last_expression[expression_level]^.operator := null_operator;
  last_expression[expression_level]^.address[1] := 7fffH;
  last_expression[expression_level]^.index := head_operand.index;
  last_expression[expression_level]^.offset := head_operand.offset;
end; { of reset_last_expression }
```

```
procedure node_display(expression : expression_pointer);
begin
  if expression <> nil then
  with expression^ do
  begin
    writeln(outfile,'(   id   : ',id);
    writeln(outfile,'    id type      : ',id_type);
    write  (outfile,'    operator     : ');
    case expression^.operator of
      addition: writeln(outfile,'+');
      division: writeln(outfile,'\');
      multiplication: writeln(outfile,'*');
      unary_minus: writeln(outfile,'u-');
      r_minus: writeln(outfile,'r-');
      unary_float: writeln(outfile,'float');
      unary_trunc: writeln(outfile,'trunc');
      unary_round: writeln(outfile,'round');
      subtraction: writeln(outfile,'-');
      or_operation:writeln(outfile,'or');
      and_operation:writeln(outfile,'and');
      unary_plus:writeln(outfile,'unary_plus');
      unary_not:writeln(outfile,'unary_not');
      greater_than:writeln(outfile,'>');
      less_than:writeln(outfile,'<');
      greater_than_or_equal:writeln(outfile,'>=');
      less_than_or_equal:writeln(outfile,'<=');
      not_equal:writeln(outfile,'<>');
      equal:writeln(outfile,'=');
      null_operator: writeln(outfile,'null');
      unary_index: writeln(outfile,'unary_index');
      otherwise writeln(outfile,expression^.operator);
    end;
    writeln(outfile,'    offset       : ',offset);
    writeln(outfile,'    index        : ',index);
    i := 1;
    while (address[i] <> 7fffH) do
    begin
      writeln(outfile,'    address[',i,'] : ',address[i]);
      i := i+1;
      if i > max_branch_pointer then
      begin
        writeln(errorfile);
        writeln(errorfile,'maximum branch pointer exceeded');
        error_found;
      end;
    end;
    if left <> nil then
      write(outfile,'    left^      : ',left^.id);
    if right <> nil then
      write(outfile,'    right^     : ',right^.id);
    if up   <> nil then
```

```
      write(outfile,'      up^      : ',up^.id);
    if down <> nil then
      write(outfile,'    down^      : ',down^.id);
    writeln(outfile,'     }');
  end
  else
    writeln(outfile,'{    Nil    }');
end;

procedure create_expression(var expression : expression_pointer);
begin
  new(expression);
  expression^.left := nil;
  expression^.right := nil;
  expression^.up := nil;
  expression^.down := nil;
  expression^.address[1] := 7fffH;
  expression^.index := blank_token;
  expression^.offset := 0;
  expression^.id_type := general_symbol_type;
  expression^.id := blank_token;
  expression^.operator := null_operator;
end;

procedure delete_expression(var expression : expression_pointer);
begin
  if expression^.up <> nil then
  begin
    expression := expression^.up;
    dispose(expression^.down);
    expression^.down := nil;
  end
  else
  begin
    if expression^.left <> nil then
    begin
      expression := expression^.left;
      dispose(expression^.right);
      expression^.right := nil;
    end
    else
      dispose(expression);
  end;
end; {of delete_expression }

function assignment_necessary(var expression : expression_pointer): boolean;
var real_zero : token_type;
begin
  str_real(0.0,real_zero);
  temp_token := blank_token;
  add_char_to_string (temp_token, '0');
```

```
  with expression^.down^.right^ do
  begin
    if ((id = temp_token) or (id = real_zero)) and
       (index = blank_token) and (offset = 0) then
    begin
      if (down = nil) and (left^.down = nil) then
        assignment_necessary := false
      else
        assignment_necessary := true;
    end
    else
       assignment_necessary := true;
  end;
end; { of assignment_necessary }

function index_assignment_necessary(var expression : expression_pointer): boolean;
var temp_id : token_type;
begin
  with expression^.down^ do
  begin
    {swap the constant to the second operand if the first operand is a constant}
  {  node_display(right); }
    if (index = blank_token) and
       ( (right^.id_type = integer_constant_symbol_type) or
         (right^.id = '0                                                  ') ) then
    begin
      if (down = nil) and (right^.down = nil) then
        if (right^.operator = addition) or (right^.operator = subtraction) then
          index_assignment_necessary := false
        else
          index_assignment_necessary := true
      else
        index_assignment_necessary := true;
    end
    else
       index_assignment_necessary := true;
  end;
end; { of index_assignment_necessary }

procedure fetch_expression_operand(expression : expression_pointer;var operand : operand_type);
begin
  operand.id := expression^.id;
  operand.id_type := expression^.id_type;
  operand.index := expression^.index;
  operand.offset := expression^.offset;
end; { fetch_expression_operand }.
```

```
File: EXPRTREE.DEF
public exprtree;
  Procedure generate_arithmetic_assignment(var expression : expression_pointer);
  Procedure generate_real_conversion(var operand:token_type);
  Procedure check_operands_type(var expression : expression_pointer);
  Procedure check_result_type(var expression : expression_pointer);
  Procedure correct_type(var expression : expression_pointer);
  Procedure type_checking(var expression : expression_pointer);
  Procedure create_arithmetic_term(var expression : expression_pointer);
  Procedure generate_arithmetic_term(var expression : expression_pointer);
  Procedure delete_term(var expression : expression_pointer);
  Procedure delete_redundant_term(var expression : expression_pointer);
  Procedure transform(var expression : expression_pointer);
  Procedure transform_right;
  Procedure transform_down;
  Procedure traverse_down;
  Procedure traverse_right;
  Procedure traverse_expression_tree;
  Procedure Transform_expression_tree;
  Procedure Evaluate_expression_tree;
```

```
File: EXPRTREE.PAS
module exprtree;
$include(exprtree.def)
$include(global.def)
$include(code_gen.def)
$include(exprsion.def)
$include(utility.def)
$include(emu_lib.def)
private exprtree;

Procedure generate_arithmetic_assignment(var expression : expression_pointer);
label 1;
var F,R,S : operand_type;
    opcode,expression_type : longint;
begin
  type_checking(expression);
  if debug then
  begin
    writeln(outfile,'---- Generate_arithmetic_assignment ----');
    node_display(expression^.left^.up);
    node_display(expression^.left);
    node_display(expression);
  end;
  if expression^.operator = unary_index then
  begin
    expression^.left^.up^.index := expression^.left^.id;
    goto 1 {exit};
  end;
  opcode := expression^.operator;
  expression_type := expression^.left^.up^.id_type;
  case expression_type of
    real_symbol_type :
    begin
      if (opcode=multiplication) or
         (opcode=addition) or
         (opcode=subtraction) or
         (opcode=unary_float) or
         (opcode=unary_plus) or
         (opcode=unary_minus) or
         (opcode=null_operator) then
      begin
        fetch_expression(F,R,S,expression);
        generate_ALU_operation(F,R,S,opcode);
      end
      else
      begin
        writeln(errorfile);
        writeln(errorfile,'operator mismatch');
        error_found;
      end;
    end;
```

```
boolean_symbol_type :
begin
  case opcode of
    greater_than : generate_greater_than(expression);
    less_than : generate_less_than(expression);
    equal : generate_equal(expression);
    not_equal : generate_not_equal(expression);
    greater_than_or_equal : generate_greater_than_or_equal(expression);
    less_than_or_equal : generate_less_than_or_equal(expression);
    unary_not : generate_unary_not(expression);
    and_operation : generate_and(expression);
    or_operation : generate_or(expression);
    addition : begin
                 fetch_expression(F,R,S,expression);
                 if (f.id[1]='#') then generate_or(expression)
                                  else generate_ALU_operation(F,R,S,opcode);
               end;
    multiplication :
               begin
                 fetch_expression(F,R,S,expression);
                 if (f.id[1]='#') then generate_or(expression)
                                  else generate_ALU_operation(F,R,S,opcode);
               end;
    otherwise write_error('unknown boolean operation                         ',blank_token);

  end;
end;
integer_symbol_type :
begin
  if (opcode=multiplication) or
     (opcode=addition) or
     (opcode=subtraction) or
     (opcode=unary_trunc) or
     (opcode=unary_round) or
     (opcode=unary_plus) or
     (opcode=unary_minus) or
     (opcode=null_operator) then
  begin
    fetch_expression(F,R,S,expression);
    generate_ALU_operation(F,R,S,opcode);
  end
  else
  begin
    writeln(errorfile);
    writeln(errorfile,'operator mismatch');
    error_found;
  end;
end;
otherwise
begin
```

```
      writeln(errorfile);
      writeln(errorfile,'unknown expression_type = ',expression_type);
      error_found;
    end;
  end; { of case expression_type of }
1: end; { of generate arithmetic assignment }

Procedure generate_real_conversion(var operand:token_type);
begin
{ expression_operator[expression_level] := unary_float;
  if operand[1] = '%' then
  begin
    assign_temp_variable(operand);
  end;
  R_operand := operand;
  str_integer(next_temp_variable_location,operand);
  temp
  F_result := concat('#',operand);
  operand := F_result;
  S_operand := blank_token;
  S_operand[1] := '0';
  expression_type := real_symbol_type;
  generate_arithmetic_microcode; }
end; { of generate_real_conversion }

Procedure check_operands_type(var expression : expression_pointer);
var real_zero : token_type;
begin
  { check to avoid real conversion when operand = 0 }
  temp_token := blank_token;
  if (expression^.id[1] = '0') and
     ((expression^.left^.id_type = real_symbol_type) or
     (expression^.left^.id_type = real_constant_symbol_type)) then
  begin
    str_real(0.0,real_zero);
    expression^.id := real_zero;
    expression^.id_type := real_constant_symbol_type;
  end
  else
  if (expression^.left^.id[1] = '0') and
     ((expression^.id_type = real_symbol_type) or
     (expression^.id_type = real_constant_symbol_type)) then
  begin
    str_real(0.0,real_zero);
    expression^.left^.id := real_zero;
    expression^.left^.id_type := real_constant_symbol_type;
  end; { of check to avoid real conversion when operand = 0 }

  if (expression^.operator=unary_minus) or
     (expression^.operator=unary_trunc) or
     (expression^.operator=unary_round) or
```

```
  (expression^.operator=unary_not) then
begin
  { acceptable }
end
else
if (expression^.operator=unary_index) then
begin
  if (expression^.left^.id_type = integer_symbol_type) or
     (expression^.left^.id_type = integer_constant_symbol_type) then
  begin
    { acceptable }
  end
  else
  begin
    writeln(errorfile);
    writeln(errorfile,'Error !!! Operands type mismatch');
    error_found;
  end;
end
else
if expression^.id_type <> expression^.left^.id_type then
begin
  if ((expression^.id_type = integer_symbol_type) or
      (expression^.id_type = integer_constant_symbol_type)) and
     ((expression^.left^.id_type = real_symbol_type) or
      (expression^.left^.id_type = real_constant_symbol_type)) then
  begin
    generate_real_conversion(expression^.id);
    expression^.id_type := real_symbol_type;
  end
  else
  if ((expression^.id_type = real_symbol_type) or
      (expression^.id_type = real_constant_symbol_type)) and
     ((expression^.left^.id_type = integer_symbol_type) or
      (expression^.left^.id_type = integer_constant_symbol_type)) then
  begin
    generate_real_conversion(expression^.left^.id);
    expression^.left^.id_type := real_symbol_type;
  end
  else
  if ((expression^.id_type = real_symbol_type) or
      (expression^.id_type = real_constant_symbol_type)) and
     ((expression^.left^.id_type = real_symbol_type) or
      (expression^.left^.id_type = real_constant_symbol_type)) then
  begin
    { acceptable combination }
  end
  else
  if ((expression^.id_type = integer_symbol_type) or
      (expression^.id_type = integer_constant_symbol_type)) and
     ((expression^.left^.id_type = integer_symbol_type) or
```

```
        (expression^.left^.id_type = integer_constant_symbol_type)) then
    begin
      { acceptable combination }
    end
    else
    if ((expression^.id_type = boolean_constant_symbol_type) or
        (expression^.id_type = boolean_symbol_type)) and
       ((expression^.left^.id_type = boolean_constant_symbol_type) or
        (expression^.left^.id_type = boolean_symbol_type)) and
       ((expression^.operator = and_operation) or
        (expression^.operator = or_operation)  or
        (expression^.operator = equal) or
        (expression^.operator= unary_not) or
        (expression^.operator = addition)) then
    begin
      { acceptable combination }
    end
    else
    begin
      writeln(errorfile);
      writeln(errorfile,'Error !!! Operands type mismatch');
      error_found;
    end;
  end;
end; {of check_operands_type }

Procedure check_result_type(var expression : expression_pointer);
begin
  with expression^ do
  begin
    if (operator=unary_trunc) or
       (operator=unary_round) then
    begin
      if (left^.up^.id[1] = '%') or (left^.up^.id[1] = '#') or
         (left^.up^.id[1] = '&') then
          left^.up^.id_type := integer_symbol_type;
      if (left^.up^.id_type = integer_symbol_type) and
        ((left^.id_type = real_symbol_type) or
           (left^.id_type = real_constant_symbol_type)) then
      begin
        { acceptable }
      end
      else
      begin
        writeln(errorfile);
        writeln(errorfile,'Error !!! type mismatch : ');
        error_found;
      end;
    end { of operator = unary_trunc .. }
    else
    if (left^.up^.id[1] = '%') or
```

```
     (left^.up^.id[1] = '#') or
     (left^.up^.id[1] = '&') then
  begin
    if (operator=greater_than) or
       (operator=less_than) or
       (operator=equal) or
       (operator=not_equal) or
       (operator=less_than_or_equal) or
       (operator=unary_not) or
       (operator=greater_than_or_equal) then
    begin
      left^.up^.id_type := boolean_symbol_type;
      expression_type := boolean_symbol_type;
    end
    else
    begin
      left^.up^.id_type := expression_type;
    end;
  end { of temporary variable '%' and '#' }
  else
  begin
    if (operator= unary_index) then
    begin
      { acceptable combination }
    end
    else
    if (left^.up^.id_type = real_symbol_type) and
      ((left^.id_type = real_symbol_type) or
       (left^.id_type = real_constant_symbol_type) or
       (left^.id_type = integer_symbol_type) or
       (left^.id_type = integer_constant_symbol_type)) then
    begin
      { acceptable combination }
    end
    else
    if (left^.up^.id_type = integer_symbol_type) and
       ((left^.id_type = integer_symbol_type) or
        (left^.id_type = integer_constant_symbol_type)) then
    begin
      { acceptable combination }
    end
    else
    if (left^.up^.id_type = boolean_symbol_type) then
    begin
      if ((left^.id_type = real_symbol_type) or
          (left^.id_type = real_constant_symbol_type) or
          (left^.id_type = integer_symbol_type) or
          (left^.id_type = integer_constant_symbol_type)) and
         ((operator=greater_than) or
          (operator=less_than) or
          (operator=equal) or
```

```
           (operator=not_equal) or
           (operator=less_than_or_equal) or
           (operator=greater_than_or_equal)) then
        begin
            expression_type := boolean_symbol_type;
        end
        else
        if ((left^.id_type = boolean_constant_symbol_type) or
            (left^.id_type = boolean_symbol_type)) and
            ((operator= and_operation) or
             (operator= or_operation)  or
             (operator= unary_not) or
             (operator= addition))  then
        begin
           { acceptable }
        end
        else
        begin
          writeln(errorfile);
          writeln(errorfile,'Error !!! type mismatch');
          error_found;
        end;
      end
      else
      begin
        writeln(errorfile);
        writeln(errorfile,'Error !!! type mismatch');
        error_found;
      end;
    end; { third case }
  end; { of with expression^ }
end; { of check_result_type }

Procedure correct_type(var expression : expression_pointer);
begin
  with expression^ do
  begin
    if id_type = real_array_symbol_type then
      id_type := real_symbol_type
    else
    if id_type = integer_array_symbol_type then
      id_type := integer_symbol_type
    else
    if id_type = boolean_array_symbol_type then
      id_type := boolean_symbol_type;
  end;
  with expression^.left^ do
  begin
    if id_type = real_array_symbol_type then
      id_type := real_symbol_type
    else
```

```
    if id_type = integer_array_symbol_type then
      id_type := integer_symbol_type
    else
    if id_type = boolean_array_symbol_type then
      id_type := boolean_symbol_type;
  end;
  with expression^.left^.up^ do
  begin
    if id_type = real_array_symbol_type then
      id_type := real_symbol_type
    else
    if id_type = integer_array_symbol_type then
      id_type := integer_symbol_type
    else
    if id_type = boolean_array_symbol_type then
      id_type := boolean_symbol_type;
  end;
end; { of correct_type }

Procedure type_checking(var expression : expression_pointer);
begin
  correct_type(expression);
  if (expression^.left^.up^.id[1] <> '#') and
     (expression^.left^.up^.id[1] <> '%') and
     (expression^.left^.up^.id[1] <> '&') then
  begin
    expression_type := expression^.left^.up^.id_type;
  end
  else
  if (expression^.id_type = real_symbol_type) or
     (expression^.id_type = real_constant_symbol_type) or
     (expression^.left^.id_type = real_symbol_type) or
     (expression^.left^.id_type = real_constant_symbol_type) then
  begin
    expression_type := real_symbol_type;
  end
  else
  if (expression^.left^.id_type=boolean_symbol_type) or
     (expression^.left^.id_type=boolean_constant_symbol_type) then
  begin
    expression_type := boolean_symbol_type;
  end
  else
  begin
    expression_type := integer_symbol_type;
  end;
  check_operands_type(expression);
  check_result_type(expression);
end; { of type_checking }

function precedence(operator : longint):longint;
```

```
begin
  case operator of
    null_operator : precedence := 0;
    greater_than  : precedence := 1;
    greater_than_or_equal : precedence := 1;
    less_than     : precedence := 1;
    less_than_or_equal    : precedence := 1;
    equal         : precedence := 1;
    not_equal     : precedence := 1;
    unary_not     : precedence := 7;
    and_operation : precedence := 6;
    or_operation  : precedence := 3;
    addition      : precedence := 3;
    subtraction   : precedence := 3;
    r_minus       : precedence := 3;
    division      : precedence := 5;
    multiplication: precedence := 6;
    unary_minus   : precedence := 7;
    unary_round   : precedence := 7;
    unary_trunc   : precedence := 7;
    unary_float   : precedence := 7;
    unary_index   : precedence := 7;
  end;
end; { of precedence }

Procedure create_arithmetic_term(var expression : expression_pointer);
var temp_expression : expression_pointer;
    token : token_type;
begin
  create_expression(new_expression);
  assign_percent_variable(token);
  new_expression^.id := token;
  new_expression^.id_type := symbol_type;
  new_expression^.operator := expression^.operator;
  expression^.operator := null_operator;
  if expression^.up <> nil then
  begin
    new_expression^.up := expression^.up;
    expression^.up := new_expression;
    new_expression^.down := expression;
    expression := new_expression^.up;
    expression^.down := new_expression;
  end
  else
  if expression^.left <> nil then
  begin
    new_expression^.down := expression;
    expression^.up := new_expression;
    new_expression^.left := expression^.left;
    expression^.left := nil;
    expression := new_expression^.left;
```

```
    expression^.right := new_expression;
  end;
  expression := new_expression^.down;
  if expression^.right^.right <> nil then
  begin
    expression := expression^.right;
    expression := expression^.right;
    expression^.left := new_expression;
    new_expression^.right := expression;
    expression := new_expression^.down;
    expression := expression^.right;
    expression^.right := nil;
  end;
  expression := new_expression;
end; { of create_arithmetic_term }

Procedure generate_arithmetic_term(var expression : expression_pointer);
begin
  if ( precedence(expression^.right^.operator) >=
       precedence(expression^.right^.right^.operator) ) then
    create_arithmetic_term(expression)
  else
  if ( expression^.right^.right^.right ) <> nil then
    generate_arithmetic_term(expression^.right)
  else
    create_arithmetic_term(expression^.right);
end; { of generate_arithmetic_term }

Procedure delete_term(var expression : expression_pointer);
begin
  if expression^.down^.down = nil then
  begin
    dispose(expression^.down);
    expression^.down := nil;
  end
  else
  begin
    new_expression := expression^.down;
    new_expression := new_expression^.down;
    new_expression^.up := expression;
    new_expression := expression^.down;
    expression^.down := new_expression^.down;
    dispose(new_expression);
  end;
end; { of delete_term }

Procedure delete_redundant_term(var expression : expression_pointer);
begin
  if (expression^.down <> nil) then
  if (expression^.down^.left = nil) then
  if (expression^.down^.right=nil) then
```

```
if (expression^.down^.operator=null_operator) then
begin
  if (expression^.down^.id[1]='%') then
  begin
    delete_term(expression);
    delete_redundant_term(expression);
  end
  else
  if (expression^.id[1]='%') then
  begin
    expression^.id := expression^.down^.id;
    expression^.id_type := expression^.down^.id_type;
    expression^.index := expression^.down^.index;
    expression^.offset := expression^.down^.offset;
    delete_term(expression);
    delete_redundant_term(expression);
  end;
end
else
if (expression^.operator = null_operator) and
   (expression^.right = nil) and
   (expression^.id[1] = '%') then
begin
  expression^.id := expression^.down^.id;
  expression^.id_type := expression^.down^.id_type;
  expression^.operator := expression^.down^.operator;
  delete_term(expression);
  delete_redundant_term(expression);
end
else
if (expression^.down^.down <> nil) then
begin
  delete_redundant_term(expression^.down);
  if (expression^.down^.down <> nil) then
  begin
    if (expression^.down^.operator = unary_not) and
       (expression^.down^.down^.operator = unary_not) then
    begin
      delete_term(expression);
      expression^.down^.operator := null_operator;
      if expression^.down^.id[1] = '%' then
        delete_term(expression);
      delete_redundant_term(expression);
    end
    else
    if (expression^.down^.operator = unary_minus) and
       (expression^.down^.down^.operator = unary_minus) then
    begin
      delete_term(expression);
      expression^.down^.operator := null_operator;
      if expression^.down^.id[1] = '%' then
```

```
          delete_term(expression);
        delete_redundant_term(expression);
      end;
    end;
  end;
end; { of delete_redundant_term }

Procedure transform(var expression : expression_pointer);
var temp_token : token_type;
begin
  if((expression^.operator = unary_minus) or
     (expression^.operator = unary_round) or
     (expression^.operator = unary_not) or
     (expression^.operator = unary_trunc) or
     (expression^.operator = unary_index)) and
     (expression^.id <> blank_token)        then
  begin
    create_expression(new_expression);
    new_expression^.id := blank_token;
    new_expression^.operator := expression^.operator;
    expression^.operator := null_operator;
    new_expression^.left := expression;
    expression^.right := new_expression;
  end
  else
  if (expression^.id[1] <> '%') and
     (expression^.id <> blank_token ) and
     (expression^.left = nil) and
     (expression^.right = nil) then
  begin
    create_expression(new_expression);
    new_expression^.id_type := expression^.id_type;
    new_expression^.id := blank_token;
    new_expression^.id[1] := '0';
    new_expression^.operator := addition;
    expression^.operator := null_operator;
    new_expression^.left := expression;
    expression^.right := new_expression;
  end;
  if (expression^.right <> nil) then
    while (expression^.right^.right <> nil) do
      generate_arithmetic_term(expression);
end; { of transform }

Procedure transform_down;
var temp_var : integer;
begin
  current_expression := current_expression^.down;
  delete_redundant_term(current_expression);
  transform(current_expression);
  if current_expression^.down <> nil then transform_down;
```

```
  if current_expression^.right <> nil then transform_right;
  if (current_expression^.index <> blank_token) then
     temp_var := assign_index(current_expression^.index);
  current_expression := current_expression^.up;
end; { of transform down }

procedure transform_right;
var temp_var : integer;
begin
  current_expression := current_expression^.right;
  delete_redundant_term(current_expression);
  transform(current_expression);
  if current_expression^.down <> nil then transform_down;
  if current_expression^.right <> nil then transform_right;
  if (current_expression^.index <> blank_token) then
     temp_var := assign_index(current_expression^.index);
  current_expression := current_expression^.left;
end; { of transform_right }

Procedure traverse_down;
begin
  current_expression := current_expression^.down;
  if current_expression^.down <> nil then traverse_down;
  if current_expression^.right <> nil then traverse_right;
{  node_display(current_expression); }
  current_expression := current_expression^.up;
  dispose(current_expression^.down);
  current_expression^.down := nil;
end; { of traverse_down }

Procedure traverse_right;
begin
  current_expression := current_expression^.right;
  if current_expression^.down <> nil then traverse_down;
  if current_expression^.right <> nil then traverse_right;
{  node_display(current_expression); }
  generate_arithmetic_assignment(current_expression);
  current_expression := current_expression^.left;
  dispose(current_expression^.right);
  current_expression^.right := nil;
end; { of traverse_right }

Procedure traverse_expression_tree;
var head_id : token_type;
    temp_var : integer;
begin
  current_expression := first_expression[expression_level];
  head_id := current_expression^.id;
  delete_redundant_term(current_expression);
  begin
    if (current_expression^.index <> blank_token) then
```

```
      temp_var := assign_index(current_expression^.index);
    transform_down;
    traverse_down;
  end;
end; { of traverse_expression_tree }

Procedure Transform_expression_tree;
var head_id : token_type;
    temp_var : integer;
begin
  current_expression := first_expression[expression_level];
  head_id := current_expression^.id;
  delete_redundant_term(current_expression);
  if (current_expression^.index <> blank_token) then
     temp_var := assign_index(current_expression^.index);
  transform_down;
end; { of transform_expression_tree }

Procedure Evaluate_expression_tree;
var head_id : token_type;
begin
  current_expression := first_expression[expression_level];
  head_id := current_expression^.id;
  traverse_down;
end; { of Evaluate_expression_tree }.
```

```
File: FETCH_TK.DEF
public fetch_tk;
   procedure fetch_token;
```

File: FETCH_TK.PAS

```
module fetch_tk;

public fetch_tk;
  procedure fetch_token;
$include (global.def)
$include (utility.def)
$include (emu_lib.def)
private fetch_tk;

procedure fetch_token;
label 1;
var
   i : longint;

procedure get_char (var ch : char);
begin
  ch := infile^;
  get (infile);
  if (ord(ch) >= 65) and (ord(ch) <= 90) then ch := char(ord(ch)+20H);
  if ord(ch) = 10 then
  begin
    gotoxy(5,24);
    write(line_number);
    line_number := line_number+1;
  end;
end;

procedure unexpected_eof;
begin
  write_error('unexpected end of file                               ',token);
end;

procedure filter_out_blank;
begin
  while ( (ord(ch) <= 20H) or (ord(ch) > 7eH) ) and ( not eof(infile) ) do
  begin
    get_char(ch);
    if not eof(infile) then write(errorfile,ch);
    if ch in upper_letter_set then ch := char(ord(ch)+20H);
  end;
end;

procedure filter_out_comment;
begin
  { filter out comments }
  if not eof(infile) then
  repeat
    get_char(ch);
    if not eof(infile) then write(errorfile,ch);
    if ch = '{' then filter_out_comment;
```

```
  until (ch = '}') or (eof(infile));
  if eof(infile) and (ch<> '}') then
  begin
    token := blank_token; token[1] := ch; { so ch can be passed as a token }
    write_error('closed comment } expected                         ',token);
  end;
  if not eof(infile) then
  begin
    get_char(ch);
    if not eof(infile) then write(errorfile,ch);
  end;
end; { of filter out comment }

procedure integer_number;
begin
  add_char_to_string (token, ch);
  if ch in ['0'..'9'] then
  begin
    if not eof(infile) then
    begin
      get_char(ch);
      if not eof(infile) then write(errorfile,ch);
    end;
    while (ch in ['0'..'9']) and (not eof(infile)) do
    begin
      add_char_to_string (token, ch);
      get_char(ch);
      if not eof(infile) then write(errorfile,ch);
    end;
  end
  else
  begin
    write_error('numeric character expected                        ',token);
  end;
end;

procedure exponent;
begin
  add_char_to_string (token, ch);
  if not eof(infile) then
  begin
    get_char(ch);;
    if not eof(infile) then write(errorfile,ch);
    if ((ch = '+') or (ch='-')) and (not eof(infile)) then
    begin
      add_char_to_string (token, ch);
      get_char(ch);;
      if not eof(infile) then write(errorfile,ch);
      integer_number;
    end
    else
```

```
    integer_number;
  end
  else
  begin
    token := blank_token; token[1] := ch; { so ch can be passed as a token }
   write_error('unexpected end of file                            ',token);
  end;
end; { of procedure exponent }

Procedure number;
begin
  symbol_type := real_constant_symbol_type;
  add_char_to_string (token, ch);
  if not eof(infile) then
  begin
    get_char(ch);;
    if not eof(infile) then write(errorfile,ch);
    if ch in ['0'..'9'] then integer_number;
    if ch = '.' then
    begin
      if not eof(infile) then
      begin
        get_char(ch);
        if not eof(infile) then write(errorfile,ch);
        if ch <> '.' then
        begin
          add_char_to_string (token, '.');
          if not eof(infile) then write(errorfile,ch);
          if ch in ['0'..'9'] then integer_number;
          if ch = 'e' then exponent;
        end
        else
          symbol_type := integer_constant_symbol_type;
      end; { of if not eof }
    end
    else
    if ch = 'e' then
      exponent
    else
      symbol_type := integer_constant_symbol_type;
  end
end; { of number }

{ expect ch : char solely to be used by procedure fetch token }

begin
  symbol_type := general_symbol_type;
  token := blank_token;
  if eof(infile) then
     begin
        add_char_to_string (token, ch);
```

```
      goto 1
    end;
  filter_out_blank;
  if ch='{' then
  begin
    repeat
      filter_out_comment;
      filter_out_blank;
    until (ch <> '{') or eof(infile);
  end;
  if (ch in ['0'..'9']) then number
  else
  if ( (ch in delimiter) or (ord(ch) < 20H) or (ord(ch) > 7eH) )
     and ( not eof(infile) ) then
  begin
    add_char_to_string (token, ch);
    if ch = '<' then
    begin
      get_char(ch);
      if ch in ['>','='] then
      begin
        add_char_to_string (token, ch);
        get_char(ch);
      end;
    end
    else
    if ch = '>' then
    begin
      get_char(ch);
      if ch = '=' then
      begin
        add_char_to_string (token, ch);
        get_char(ch);
      end;
    end
    else
      get_char(ch);
    if not eof(infile) then write(errorfile,ch);
  end
  else
  begin
    add_char_to_string (token, ch);
    if not eof(infile) then
      repeat
        get_char(ch);
        if not eof(infile) then write(errorfile,ch);
        if not ( (ch in delimiter) or (ord(ch) < 20H)
                   or (ord(ch) > 7eH) ) then
           add_char_to_string (token, ch);
      until (ch in delimiter) or (ord(ch) < 20H)
           or (ord(ch) > 7eH) or  eof(infile);
```

```
  end;
  if symbol_type = real_constant_symbol_type then
  begin
    val_real (token,real_constant_value,i);
    str_real (real_constant_value,token);
    if i <> 0 then
    begin
      write_error('real conversion error                              ',token);
    end;
  end
  else
  if symbol_type = integer_constant_symbol_type then
  begin
    val_integer (token,integer_constant_value,i);
    str_integer (integer_constant_value,token);
  if i <> 0 then
    begin
      write_error('real conversion error                              ',token);
    end
  end;
1: end; { of fetch_token }.
```

File: FOR_STAT.PAS

```
procedure for_loop(for_index,upper:operand_type);
var begin_loop_address : longint;
    temp_operand : operand_type;
    one : operand_type;
begin
  clear_pipeline_stage;
  { check for exit condition }
  assign_stack_operand(temp_operand,integer_symbol_type);
  free_stack_operand;
  temp_operand.offset := 0;
  temp_operand.index := blank_token;
  branch_lookahead_buffer[2] := if_negative;
  begin_loop_address := program_counter;
  generate_ALU_operation(temp_operand,upper,for_index,subtraction);
  compound_statement;
  { increment for_index }
  one.id := blank_token;
  one.id[1] := '1';
  one.id_type := integer_constant_symbol_type;
  one.offset := 0;
  one.index := blank_token;
  generate_ALU_operation(for_index,for_index,one,addition);
  if (write_lookahead_buffer[1].id <> blank_token) then generate_nop;
  writeln(outfile,';',program_counter,': goto ',begin_loop_address);
  microcode_address := program_counter;
  branch_address := begin_loop_address;
  AM2910_opcode := CJP;
  branch_opcode := unconditional;
  output_microcode_field;
  program_counter := program_counter + 1;
  writeln(outfile,'b ',begin_loop_address+2,' ',program_counter);
end; { of for_loop }

procedure for_statement;
var for_index,upper : operand_type;
    begin_loop_address,lower_offset, upper_offset: longint;
    temp_token : token_type;
    i : longint;
    initial_index_register : array[0..max_index_register] of token_type;
begin
{  writeln(outfile,';-------- expression ',expression_number,' --------'); }
  expression_number := expression_number + 1;
  fetch_token;
  find_symbol(token,symbol_type,symbol_value,found);
  if symbol_type <> integer_symbol_type then
  begin
    write_error('integer expected                                    ',token);
  end;
  for_index.id := token;
  for_index.id_type := integer_symbol_type;
```

```
  for_index.offset := 0;
  for_index.index := blank_token;
  fetch_token;
  verify_token(token,colon);
  fetch_token;
  verify_token(token,equal_token);
  fetch_assigned_parameter(for_index);
  verify_token(token,to_token);
  upper.id_type := integer_symbol_type;
  fetch_parameter(upper);
  { begin of the for-loop }
  for i := 0 to max_index_register do
  begin
    initial_index_register[i] := index_register[i];
  end;
  for_loop(for_index,upper);
  {restore the index register to its initial state}
  for i := 0 to max_index_register do
  begin
    if initial_index_register[i] <> index_register[i] then
    begin
      if initial_index_register[i] <> blank_token then
        load_index_register(initial_index_register[i]);
      index_register[i] := initial_index_register[i];
    end;
  end;
end; { of for_statement }
```

```
File: GLOBAL.DEF
public global;
const
  pi = 3.141592654;
  prime = 19;
  max_token_length = 50;
  dataram_address_limit = 2047;
  program_counter_limit = 4095;
  temp_variable_limit = 100; { used to store temporary variables }
  max_branch_pointer = 20;
  max_index_register = 15;
  max_procedure_level = 6;
  max_reference_parameter = 15;
  max_expression_level = 10;
  max_local_variable = 100;
  version =          'Computer Architecture Lab Pascal Compiler - 2.0   ';
  program_heading  = 'program                                           ';
  procedure_heading= 'procedure                                         ';
  function_heading = 'function                                          ';
  blank_token      = '                                                  ';
  semicolon        = ';                                                 ';
  open_parenthesis = '(                                                 ';
  close_parenthesis= ')                                                 ';
  comma            = ',                                                 ';
  colon            = ':                                                 ';
  greater_than_or_equal_token
                   = '>=                                                ';
  less_than_or_equal_token
                   = '<=                                                ';
  greater_than_token
                   = '>                                                 ';
  less_than_token  = '<                                                 ';
  not_equal_token  = '<>                                                ';
  equal_token      = '=                                                 ';
  not_token        = 'not                                               ';
  and_token        = 'and                                               ';
  or_token         = 'or                                                ';
  multiply_token   = '*                                                 ';
  divide_token     = '/                                                 ';
  add_token        = '+                                                 ';
  plus_token       = '+                                                 ';
  percent_token    = '%                                                 ';
  minus_token      = '-                                                 ';
  period           = '.                                                 ';
  open_bracket     = '{                                                 ';
  close_bracket    = '}                                                 ';
  var_declaration  = 'var                                               ';
  const_declaration= 'const                                             ';
  type_declaration = 'type                                              ';
  begin_token      = 'begin                                             ';
  do_token         = 'do                                                ';
```

```
to_token            = 'to                                  ';
for_token           = 'for                                 ';
end_token           = 'end                                 ';
if_token            = 'if                                  ';
then_token          = 'then                                ';
else_token          = 'else                                ';
while_token         = 'while                               ';
repeat_token        = 'repeat                              ';
until_token         = 'until                               ';
true_token          = 'true                                ';
false_token         = 'false                               ';
real_token          = 'real                                ';
integer_token       = 'integer                             ';
boolean_token       = 'boolean                             ';
round_token         = 'round                               ';
gt_ffs_token        = 'gt_ffs                              ';
trunc_token         = 'trunc                               ';
exp_token           = 'exp                                 ';
ln_token            = 'ln                                  ';
sqrt_token          = 'sqrt                                ';
sin_token           = 'sin                                 ';
cos_token           = 'cos                                 ';
tan_token           = 'tan                                 ';
asin_token          = 'asin                                ';
acos_token          = 'acos                                ';
atan_token          = 'atan                                ';
send                = 'send                                ';
send_msw            = 'send_msw                            ';
send_lsw            = 'send_lsw                            ';
receive             = 'receive                             ';
receive_msw         = 'receive_msw                         ';
receive_lsw         = 'receive_lsw                         ';
proc_reset          = 'initialize                          ';
store_function      = 'store_function                      ';
store_window        = 'store_window                        ';
read_function       = 'read_function                       ';
network             = 'network                             ';
host                = 'host                                ';
host_dav            = 'host_dav                            ';
host_rfi            = 'host_rfi                            ';
network_dav         = 'network_dav                         ';
network_rfi         = 'network_rfi                         ';
real_zero           = '0.0                                 ';
integer_zero        = '0                                   ';
array_token         = 'array                               ';
of_token            = 'of                                  ';
open_square_bracket  = '[                                      ';
close_square_bracket = ']                                      ';
dot                  = '.                                      ';

{symbol type}
```

```
real_symbol_type                 =0;
integer_symbol_type              =1;
real_constant_symbol_type        =2;
integer_constant_symbol_type     =3;
label_symbol_type                =4;
procedure_symbol_type            =5;
standard_procedure_symbol_type   =6;
function_symbol_type             =7;
standard_function_symbol_type    =8;
reserved_word_symbol_type        =9;
delimiter_symbol_type            =10;
boolean_symbol_type              =11;
boolean_constant_symbol_type     =12;
real_array_symbol_type           =13;
integer_array_symbol_type        =14;
boolean_array_symbol_type        =15;
general_symbol_type              =100;

{ parameter type }
call_by_reference = 0;
call_by_value = 1;

{operator type}
addition =0;
multiplication =1;
subtraction =2;
unary_minus = 3;
unary_round =4;
unary_trunc =5;
division =6;
unary_float = 7;
r_minus = 8;
greater_than=9;
greater_than_or_equal=10;
less_than=11;
less_than_or_equal=12;
equal=13;
not_equal=14;
unary_not=15;
and_operation=16;
or_operation=17;
unary_plus=18;
unary_index=19;
null_operator = 100;

{assignment type}
R_type = 2;
F_type = 3;
M_type = 4;  { temporary variable type }

{branch type}
```

```
if_not_xdav = 0; { if network has data available }
if_not_xrfi = 1; { if network is ready to receive data }
if_not_hdav = 2; { if host port has data available }
if_not_hrfi = 3; { if host port is ready to receive data }
if_zero = 4; { if ALU result is zero }
if_not_zero = 5; { if ALU result is not zero }
if_negative = 6; { if ALU result is negative }
if_not_negative = 9; { if ALU result is not negative }
if_error = 10; { if error occurrs }
unconditional = 12; { unconditional branch }
no_branch = 7; { force condition to be always false }

{am2910 opcode}
JZ = 0; { jump to zero }
CJS = 1; { conditonal jump to subroutine }
CJP = 3; { conditional jump }
CRTN = 10;  { conditional return }
CONT = 14; { continue }

{write_opcode}
write_ALU = 1;
write_host = 2;
write_network = 3;
read_function_opcode = 4;
store_window_opcode = 6;
store_function_opcode = 7;

{ALU_opcode}
Float_add = 0;
Float_sub = 1;
Float_mult = 2;
Float_convert = 5;
Fix_convert = 4;

{read_opcode}
load_index = 1;  { store a value to the index register from the R-bus }
load_host = 2;   { store a value to the host fifo from the R_bus }
load_network = 3; { store a value to the network fifo to the R_bus }


type
  delimiter_type = set of char;
  Upper_letter_set_type = set of char;
  token_type = packed array [1..max_token_length] of char;
  character_set = set of char;
  symbol_pointer = ^symbol_record;
  parameter_pointer = ^parameter_record;
  parameter_record =
  record
    id : token_type;
```

```
    id_type : longint;
    address : longint;
    parameter_type : longint;
    next : parameter_pointer;
  end;
  symbol_record =
  record
    name    : token_type;
    value   : longint;
    symbol_type : longint;
    scope : longint;
    constant_value: real;
    parameter_link : parameter_pointer;
    next    : symbol_pointer;
  end;
  expression_pointer = ^expression_record;
  expression_record =
  record
    id : token_type;
    id_type : longint;
    operator   : longint;
    offset   : longint;
    index : token_type;
    address : array[0..max_branch_pointer] of longint;
    up,down,left,right: expression_pointer;
  end;
  operand_type =
  record
    id : token_type;
    index : token_type;
    offset : longint;
    id_type : longint;
    id_address : longint;
    index_address : longint;
  end;
var
  delimiter : delimiter_type;
  Upper_letter_set : Upper_letter_set_type;
  token : token_type;
  infile : file of char;
  outfile,errorfile,constant_file : text;
  ch : char; { this is to be used only by procedure fetch_token }
  i : longint;
  real_constant_value : real;
  integer_constant_value : longint;
  new_symbol,current_symbol : symbol_pointer;
  first_symbol : array[0..prime] of symbol_pointer;
  symbol_array : array [0..30] of token_type;  { used to store multiple var declaration }
  program_counter,dataram_address,symbol_type,symbol_value : longint;
  constant_program_counter : longint;
  temp_variable_address : longint;
```

```
constant_assignment_type : longint;
symbol_name : token_type;
found : boolean;
branch_opcode,am2910_opcode,branch_address,write_opcode,read_opcode : longint;
microcode_address,AR,AS,IA1,IA0,Dsel :longint;
AIR,AIS : array[0..1] of longint;
AIF,IA2 : array[0..2] of longint;
AF : array[0..2] of longint;
I3 : array[0..1] of longint;
ENF_bar,I4,msw : longint;
mc325_buffer : array [0..1] of longint;
first_expression,last_expression : array[0..max_expression_level] of expression_pointer;
new_expression,current_expression: expression_pointer;
write_lookahead_buffer : array [0..2] of operand_type;
branch_lookahead_buffer : array [0..2] of longint;
{ used for pipeline stage of am29325. Need to stuff nop's when
  id needed for calculation has not been calculated for at least
  two cycles previously }
line_number,percent_variable_counter : longint;
hex_number : token_type;
input_filename : token_type;
output_filename :token_type;
error_filename : token_type;
constant_filename : token_type;
filename : token_type;
expression_type : longint;
expression_operator : array[ 0..max_expression_level] of longint;
branch_state : longint; { 1 = branch if condition is true
                          2 = branch if condition is false
                          3 = true and false address jump may be required
                              ( last boolean expression evaluated )
                          4 = branch if condition is true
                          5 = branch if condition is false }
expression_number : longint; { number used to keep track of the nummber of
                               expressions }
array_lower_range, array_upper_range : longint;
debug : boolean;
index_register : array[0..max_index_register] of token_type;
stack_pointer : longint;
index_pointer : longint;
procedure_link : symbol_pointer;
current_parameter,new_parameter : parameter_pointer;
procedure_level,parameter_type : longint;
local_variable : array[0..max_procedure_level,0..max_local_variable] of token_type;
inside_function_block : array[0..max_procedure_level] of boolean;
no_local_variable : array[0..max_procedure_level] of longint;
expression_level : longint;
operand : operand_type;
zero_operand : operand_type;
temp_token : token_type;
```

```
File: GLOBAL.PAS
module global;
$include(global.def)
private global
{ This module is one of the many datums showing that: }
{ Intel PASCAL86 is quintessentially bogus };.
```

```
File: HEX_CONV.DEF
public hex_conv;
  procedure word_to_hex(number : word; var hex_number : token_type);
  Function Hex_to_word(hex_number : token_type):word;
  procedure byte_to_hex(number : word; var hex_number : token_type);
  Function Hex_to_byte(hex_number : token_type):word;
  procedure integer_to_hex(number : longint; var hex_number : token_type);
  Function Hex_to_integer(hex_number : token_type):longint;
  Function Hex(data:word):char;
```

```
File: HEX_CONV.PAS
module hex_conv;

$include (hex_conv.def)
$include (emu_lib.def)
$include (bit_func.def)
$include (global.def)

private hex_conv;
  Function Hex(data:word):char;
  var i : longint;
  begin
    case word_and (data, 000fH) of
      10 : hex := 'A';
      11 : hex := 'B';
      12 : hex := 'C';
      13 : hex := 'D';
      14 : hex := 'E';
      15 : hex := 'F';
      otherwise hex := chr(48 + i);
    end;
  end; { Hex }

  procedure byte_to_hex(number : word; var hex_number : token_type);
  var p, i :word;
  begin
    hex_number := blank_token;
    for p := 1 to 2 do
       hex_number[p] := '0';
    for i := 0 to 1 do
     hex_number[2-i] := hex(word_shr (number,(4*i)))
  end; { of byte_to_hex }

  procedure word_to_hex(number : word; var hex_number :token_type);
  var p, i : word;
  begin
    hex_number := blank_token;
    for p := 1 to 4 do
     hex_number[p] := '0';
    for i := 0 to 3 do
     hex_number[4-i] := hex(word_shl (number,(4*i)))
  end; { of word_to_hex }

  procedure integer_to_hex(number : longint; var hex_number : token_type);
  var p, i : word;
  begin
    hex_number := blank_token;
    for p := 1 to 8 do
     hex_number[p] := '0';
    for i := 0 to 7 do
    hex_number[8-i] := hex(dword_shr (number,(4*i)))
```

```
end; { of integer_to_hex }

function Hex_to_byte(hex_number : token_type):word;
label 1;
var i,j,m :word;
    returned_byte : word;
begin
  j := 0;
  for i := 1 to length(hex_number) do
  begin
    if hex_number[i] in ['A'..'F'] then
      hex_number[i] := char(ord(hex_number[i])+32);
    if (hex_number[i] in ['0'..'9']) or (hex_number[i] in ['a'..'f']) then
      j := j + 1
    else
    if (hex_number[i] <> 'H') and (hex_number[i] <> ' ') then
    begin
      writeln (' invalid character  : ',hex_number[i],' in hex number ', hex_number);
        goto 1; { exit }
    end; { of invalid hex character }
  end; { for i .. }
  returned_byte := 0;
  for i := 1 to j do
  begin
    if hex_number[i] in ['0'..'9'] then
      m := ord(hex_number[i]) -48
    else
      m := ord(hex_number[i]) - 87;
    returned_byte := word_or (returned_byte, (word_shl (m, (4*(j-i)))));
  end; { for i ... }
  Hex_to_byte := returned_byte;
1: end; { Function Hex_to_byte }

Function Hex_to_word(hex_number : token_type):word;
label 1;
var i,j,m : word;
    returned_word : word;
begin
  j := 0;
  for i := 1 to length(hex_number) do
  begin
    if hex_number[i] in ['A'..'F'] then
      hex_number[i] := char(ord(hex_number[i])+32);
    if (hex_number[i] in ['0'..'9']) or (hex_number[i] in ['a'..'f']) then
      j := j + 1
    else
    if (hex_number[i] <> 'H') and (hex_number[i] <> ' ') then
    begin
      writeln (' invalid character  : ',hex_number[i],' in hex number ', hex_number);
        goto 1  { exit }
    end; { of invalid hex character }
```

```
    end; { for i .. }
    returned_word := 0;
    for i := 1 to j do
    begin
      if hex_number[i] in ['0'..'9'] then
        m := ord(hex_number[i]) -48
      else
        m := ord(hex_number[i]) - 87;
      returned_word := word_or(returned_word, (word_shl (m, (4*(j-i)))));
    end; { for i ... }
    Hex_to_word := returned_word;
1: end; { Function Hex_to_word }

  Function Hex_to_integer(hex_number : token_type):longint;
  label 1;
  var i,j,m : longint;
      returned_integer : longint;
  begin
    j := 0;
    for i := 1 to length(hex_number) do
    begin
      if hex_number[i] in ['A'..'F'] then
        hex_number[i] := char(ord(hex_number[i])+32);
      if (hex_number[i] in ['0'..'9']) or (hex_number[i] in ['a'..'f']) then
        j := j + 1
      else
      if (hex_number[i] <> 'H') and (hex_number[i] <> ' ') then
      begin
        writeln (' invalid character  : ',hex_number[i],' in hex number ', hex_number);
          goto 1;  { exit }
      end; { of invalid hex character }
    end; { for i .. }
    returned_integer := 0;
    for i := 1 to j do
    begin
      if hex_number[i] in ['0'..'9'] then
        m := ord(hex_number[i]) -48
      else
        m := ord(hex_number[i]) - 87;
      returned_integer := dword_or (returned_integer, (dword_shl (m, (4*(j-i)))));
    end; { for i ... }
    Hex_to_integer := returned_integer;
1: end; { Function Hex_to_integer }.
```

File: IEEE_CNV.DEF

```
public ieee_cnv;
  procedure real_to_ieee(value:real;var msw,lsw:word);
  function ieee_to_real(msw,lsw:word):real;
  function longint_to_real(lw : longint):real;
  function real_to_longint(sx : real):longint;
```

File: IEEE_CNV.PAS

```
module ieee_cnv;
$include (bit_func.def)
$include (ieee_cnv.def)

private ieee_cnv;
  type
    real_internal_type = record
                       case internal_type: boolean of
                        true : (real_value : real);
                        false : (word_value : array [1..2] of word);
                       end;

procedure real_to_ieee;
var
  x : real_internal_type;
begin
  x.real_value := value;
  msw := x.word_value[2];
  lsw := x.word_value[1];
end;

function ieee_to_real;
var
  x : real_internal_type;
begin
  x.word_value[2] := msw;
  x.word_value[1] := lsw;
  ieee_to_real := x.real_value;
end;

function longint_to_real(lw : longint):real;
var
  x : real_internal_type;
begin
  x.word_value[2] := dword_shr(lw,16);
  x.word_value[1] := dword_and(lw,0000ffffH);
  longint_to_real := x.real_value;
end; { of longint_to_real }

function real_to_longint(sx : real):longint;
var
  x : real_internal_type;
  lw : longint;
begin
  x.real_value := sx;
  lw := x.word_value[2];
  lw := dword_or(dword_shl(lw,16),x.word_value[1]);
  real_to_longint := lw;
end;. { of real_to_longint }
```

```
File: IF_WHILE.PAS
procedure reverse_branch_direction(var address : longint);
begin
  if (branch_lookahead_buffer[1]=if_zero) then
    branch_lookahead_buffer[1]:=if_not_zero
  else
  if (branch_lookahead_buffer[1]=if_not_zero) then
    branch_lookahead_buffer[1]:= if_zero
  else
  if (branch_lookahead_buffer[1]=if_negative) then
    branch_lookahead_buffer[1]:=if_not_negative
  else
  if (branch_lookahead_buffer[1]=if_not_negative) then
    branch_lookahead_buffer[1]:=if_negative;
  address := -address;
end; { of reverse_branch_direction }

Procedure while_statement;
var address : array [1..max_branch_pointer] of longint;
    starting_address,i : longint;
    initial_index_register : array[0..max_index_register] of token_type;
begin
  fetch_token;
  clear_pipeline_stage;
  starting_address := program_counter;
  for i := 0 to max_index_register do
  begin
    initial_index_register[i] := index_register[i];
  end;
  boolean_expression;
  { store the branch address before another compound expression is called }
  i := 1;
  repeat
    address[i] := first_expression[expression_level]^.address[i];
    i := i+1;
    if i > max_branch_pointer then
      write_error('maximum branch pointer exceeded                    ',token);
  until (address[i-1] = 7fffh);
  if (i>2) then
    if (address[i-2] > 0) then reverse_branch_direction(address[i-2]);
  { back patching branch address with true condition }
  i := 1;
  repeat
    if (address[i] > 0) and (address[i] <> 7fffh) then
    begin
      clear_pipeline_stage;
      writeln(outfile,'b ',address[i]+2,' ',program_counter);
    end;
    i := i+1;
    if i > max_branch_pointer then
      write_error('maximum branch pointer exceeded                    ',token);
```

```
  until (address[i-1] = 7fffh);
  verify_token(token,do_token);
  compound_statement;
  {restore the index register to its initial state}
  for i := 0 to max_index_register do
  begin
    if initial_index_register[i] <> index_register[i] then
    begin
      if initial_index_register[i] <> blank_token then
        load_index_register(initial_index_register[i]);
      index_register[i] := initial_index_register[i];
    end;
  end;
  if write_lookahead_buffer[1].id <> blank_token then generate_Nop;
  writeln(outfile,';',program_counter,': goto ',starting_address);
  microcode_address := program_counter;
  branch_address := starting_address;
  AM2910_opcode := CJP;
  branch_opcode := unconditional;
  output_microcode_field;
  program_counter := program_counter + 1;
  i := 1;
  while (address[i] <> 7fffh) do
  begin
    if address[i] < 0 then
       writeln(outfile,'b ',abs(address[i])+2,' ',program_counter);
    i := i+1;
    if i > max_branch_pointer then
      write_error('maximum branch pointer exceeded                    ',token);
  end;
end; { of while_statement }

Procedure if_statement;
var   address : array [1..max_branch_pointer] of longint;
      endif_address,else_address,i : longint;
      initial_index_register : array[0..max_index_register] of token_type;
begin
  fetch_token;
  boolean_expression;
  { store the branch address before another compound expression is called }
  i := 1;
  repeat
    address[i] := first_expression[expression_level]^.address[i];
    i := i+1;
    if i > max_branch_pointer then
      write_error('maximum branch pointer exceeded                    ',token);
  until (address[i-1] = 7fffh);
  if (i>2) then
    if (address[i-2] > 0) then reverse_branch_direction(address[i-2]);
  { back patching branch address with true condition }
  i := 1;
```

```
repeat
  if (address[i] > 0) and (address[i] <> 7fffh) then
  begin
    clear_pipeline_stage;
    writeln(outfile,'b ',address[i]+2,' ',program_counter);
  end;
  i := i+1;
  if i > max_branch_pointer then
    write_error('maximum branch pointer exceeded                    ', token);
until (address[i-1] = 7fffh);
verify_token(token,then_token);
for i := 0 to max_index_register do
begin
  initial_index_register[i] := index_register[i];
end;
compound_statement;
{restore the index register to its initial state}
for i := 0 to max_index_register do
begin
  if initial_index_register[i] <> index_register[i] then
  begin
    if initial_index_register[i] <> blank_token then
      load_index_register(initial_index_register[i]);
    index_register[i] := initial_index_register[i];
  end;
end;
if token = else_token then
begin
  if write_lookahead_buffer[1].id <> blank_token then generate_Nop;
  endif_address := program_counter;
  am2910_opcode := CJP;
  branch_lookahead_buffer[0] := unconditional;
  microcode_address := program_counter;
  output_microcode_field;
  program_counter := program_counter + 1;
  else_address := program_counter;
  compound_statement;
  {restore the index register to its initial state}
  for i := 0 to max_index_register do
  begin
    if initial_index_register[i] <> index_register[i] then
    begin
      if initial_index_register[i] <> blank_token then
        load_index_register(initial_index_register[i]);
      index_register[i] := initial_index_register[i];
    end;
  end;
  clear_pipeline_stage;
  writeln(outfile,'b ',endif_address,' ',program_counter);
end
else
```

```
  begin
    clear_pipeline_stage;
    else_address := program_counter;
  end;
  i := 1;
  while (address[i] <> 7fffh) do
  begin
    if address[i] < 0 then
       writeln(outfile,'b ',abs(address[i])+2,' ',else_address);
    i := i+1;
    if i > max_branch_pointer then
      write_error('maximum branch pointer exceeded                   ',token);
  end;
end; { of if_statement }

Procedure repeat_statement;
var address : longint;
    starting_address,i : longint;
    initial_index_register : array[0..max_index_register] of token_type;
    index_pointer : longint;
begin
  clear_pipeline_stage;
  starting_address := program_counter;
  { save the state of the index register }
  for i := 0 to max_index_register do
  begin
    initial_index_register[i] := index_register[i];
  end;
  compound_statement;
  while token[1]=';' do
  begin
    compound_statement;
  end;
  verify_token(token,until_token);
  fetch_token;
  boolean_expression;
  { back patching branch address }
  i := 1;
  repeat
    if (first_expression[expression_level]^.address[i] <> 7fffh) then
    begin
      if (first_expression[expression_level]^.address[i+1]=7fffh) and
         (first_expression[expression_level]^.address[i]>0) then
      begin
        address := first_expression[expression_level]^.address[i];
        reverse_branch_direction(address);
        first_expression[expression_level]^.address[i] := address;
      end;
      if (first_expression[expression_level]^.address[i] < 0) then
        writeln(outfile,'b ',-first_expression[expression_level]^.address[i]+2,' ',starting_address)
      else
```

```
      writeln(outfile,'b ',first_expression[expression_level]^.address[i]+2,' ',program_counter)
    end;
    i := i+1;
    if i > max_branch_pointer then
      write_error('maximum branch pointer exceeded                   ',token);
  until (first_expression[expression_level]^.address[i-1] = 7fffh);
  {restore the index register to its initial state}
  for i := 0 to max_index_register do
  begin
    if initial_index_register[i] <> index_register[i] then
    begin
      if initial_index_register[i] <> blank_token then
        load_index_register(initial_index_register[i]);
      index_register[i] := initial_index_register[i];
    end;
  end;
end; { of repeat_statement }
```

```
File: INIT.DEF
public init;
type
   pac81 = packed array [1..81] of char;

  procedure initialize;
  procedure program_heading_block;
  procedure insert_std_procedure_to_symbol_table;
  procedure insert_standard_function_to_symbol_table;
  procedure insert_reserved_word_to_symbol_table;
  procedure insert_delimiter_to_symbol_table;
```

```
File: INIT.PAS
module init;

$include(init.def)
$include(global.def)
$include(fetch_tk.def)
$include(symbol_t.def)
$include(utility.def)
$include(code_gen.def)
$include(exprsion.def)
$include(emu_lib.def)
public UDIcall;
   function dqgetargument ( var string81 : pac81; var w : word) :char;
   function dqattach ( var token : token_type; var error : word) : word;
   procedure dqdetach (connection : word; var error : word);
   procedure dqexit (completion_code : word);

private init;

procedure insert_std_procedure_to_symbol_table;
begin
  symbol_type := standard_procedure_symbol_type;
  insert_symbol(send,symbol_type,symbol_value);
  insert_symbol(send_msw,symbol_type,symbol_value);
  insert_symbol(send_lsw,symbol_type,symbol_value);
  insert_symbol(receive,symbol_type,symbol_value);
  insert_symbol(receive_msw,symbol_type,symbol_value);
  insert_symbol(receive_lsw,symbol_type,symbol_value);
  insert_symbol(store_function,symbol_type,symbol_value);
  insert_symbol(store_window,symbol_type,symbol_value);
  insert_symbol(read_function,symbol_type,symbol_value);
  insert_symbol(proc_reset,symbol_type,symbol_value);
  insert_symbol(gt_ffs_token,symbol_type,symbol_value);
end;

procedure insert_standard_function_to_symbol_table;
begin
  symbol_type := standard_function_symbol_type;
  insert_symbol(trunc_token,symbol_type,symbol_value);
  insert_symbol(round_token,symbol_type,symbol_value);
  insert_symbol(exp_token,symbol_type,symbol_value);
  insert_symbol(ln_token,symbol_type,symbol_value);
  insert_symbol(sqrt_token,symbol_type,symbol_value);
  insert_symbol(sin_token,symbol_type,symbol_value);
  insert_symbol(cos_token,symbol_type,symbol_value);
  insert_symbol(tan_token,symbol_type,symbol_value);
  insert_symbol(asin_token,symbol_type,symbol_value);
  insert_symbol(acos_token,symbol_type,symbol_value);
  insert_symbol(atan_token,symbol_type,symbol_value);
end;
```

```
procedure insert_reserved_word_to_symbol_table;
begin
  symbol_type := reserved_word_symbol_type;
  insert_symbol(program_heading,symbol_type,symbol_value);
  insert_symbol(procedure_heading,symbol_type,symbol_value);
  insert_symbol(function_heading,symbol_type,symbol_value);
  insert_symbol(var_declaration,symbol_type,symbol_value);
  insert_symbol(const_declaration,symbol_type,symbol_value);
  insert_symbol(type_declaration,symbol_type,symbol_value);
  insert_symbol(begin_token,symbol_type,symbol_value);
  insert_symbol(end_token,symbol_type,symbol_value);
  insert_symbol(if_token,symbol_type,symbol_value);
  insert_symbol(then_token,symbol_type,symbol_value);
  insert_symbol(else_token,symbol_type,symbol_value);
  insert_symbol(while_token,symbol_type,symbol_value);
  insert_symbol(repeat_token,symbol_type,symbol_value);
  insert_symbol(until_token,symbol_type,symbol_value);
  insert_symbol(do_token,symbol_type,symbol_value);
  insert_symbol(real_token,symbol_type,symbol_value);
  insert_symbol(integer_token,symbol_type,symbol_value);
  insert_symbol(boolean_token,symbol_type,symbol_value);
  insert_symbol(host,symbol_type,symbol_value);
  insert_symbol(network,symbol_type,symbol_value);
  insert_symbol(greater_than_token,symbol_type,symbol_value);
  insert_symbol(greater_than_or_equal_token,symbol_type,symbol_value);
  insert_symbol(less_than_token,symbol_type,symbol_value);
  insert_symbol(less_than_or_equal_token,symbol_type,symbol_value);
  insert_symbol(not_equal_token,symbol_type,symbol_value);
  insert_symbol(equal_token,symbol_type,symbol_value);
  insert_symbol(not_token,symbol_type,symbol_value);
  insert_symbol(and_token,symbol_type,symbol_value);
  insert_symbol(or_token,symbol_type,symbol_value);
  insert_symbol(array_token,symbol_type,symbol_value);
  insert_symbol(of_token,symbol_type,symbol_value);
  insert_symbol(to_token,symbol_type,symbol_value);
  insert_symbol(for_token,symbol_type,symbol_value);
end;

procedure insert_delimiter_to_symbol_table;
begin
  symbol_type := delimiter_symbol_type;
  insert_symbol(multiply_token,symbol_type,symbol_value);
  insert_symbol(divide_token,symbol_type,symbol_value);
  insert_symbol(add_token,symbol_type,symbol_value);
  insert_symbol(minus_token,symbol_type,symbol_value);
  insert_symbol(period,symbol_type,symbol_value);
  insert_symbol(comma,symbol_type,symbol_value);
  insert_symbol(percent_token,symbol_type,symbol_value);
  insert_symbol(open_parenthesis,symbol_type,symbol_value);
  insert_symbol(close_parenthesis,symbol_type,symbol_value);
  insert_symbol(semicolon,symbol_type,symbol_value);
```

```
  insert_symbol(open_bracket,symbol_type,symbol_value);
  insert_symbol(close_bracket,symbol_type,symbol_value);
  insert_symbol(close_square_bracket,symbol_type,symbol_value);
  insert_symbol(open_square_bracket,symbol_type,symbol_value);
end;

procedure initialize;
var i,j : longint;
    string81 : pac81;  {     These three variables are used by    }
    w : word;          { udi system call DOGETARGUMENT. See page }
    c : char;          { 32 of UDI System Calls IRMX Vol. 2       }

    connection,error : word; {     Used by dqattach and dqdetach. See pg.
                              12 & 25 of UDI System Calls IRMX Vol. 2   }

    completion_code : word;  {     Used by dqexit. See pg. 26 UDI System
                              Calls IRMX Vol. 2                         }

begin
 {      Dqgetargument emulates turbo pascal's ability to get parameters off
   of the command line. See pg. 32 of UDI System Calls IRMX Vol. 2          }

  c := dqgetargument(string81,w);
  for i := 0 to prime do
  begin
    first_symbol[i] := nil;
  end;
  procedure_level := 0;
  clearscreen;
  gotoxy(5,2);
  writeln(version);
  gotoxy(5,5);
  write('input file : ');
  filename := blank_token;

{     This block of code enables the user to type in the name of the file
   with or without the .pas extention.                                    }

  if ord(string81[1]) = 0 then
  begin
    read_token(temp_token);
    i := 1;
    while (i <= max_token_length) and (temp_token[i] <> '.') do
    begin
      filename[i] := temp_token[i];
      i := i + 1
    end
  end
  else
  begin
    i := 2;
```

```
    while (string81[i] <> '.') and (i <= (ord(string81[1])) + 1) and
                                                (i <= max_token_length) do
    begin
      filename[i-1] := string81[i];
      i := i + 1
    end;
    write_token(output, filename)
  end;

  debug := false;
  gotoxy(5,24);
  write(0);
  line_number := 1;
  input_filename := filename;
  concat (input_filename, '.pas                                             ');
  output_filename := filename;
  concat (output_filename, '.fpp                                             ');
  error_filename := filename;
  concat (error_filename, '.err                                             ');
  constant_filename := filename;
  concat (constant_filename, '.hct                                             ');

{     Temp_token is manipulated like so in the next seven lines because
  the UDI call dqattach uses a different string format than Pascal-86.
  Dqattach requires that the string length be stored in string[1]         }

  temp_token := input_filename;
  j := length(temp_token);
  if j = max_token_length then
    j := j - 1;
  for i := j downto 1 do
    temp_token[i+1] := temp_token[i];
  temp_token[1] := chr(j);

{     Dqattach is used for an indirect pupose here, which is to detect
  whether or not the file exists. If it does it detaches the file with
  dqdetach (so the file is its original state be for dqattach was called).
  If the file does not exist, the error handling occurs here, not later with
  a cryptic Pascal-86 message. See pp. 12 & 25 in UDI System Calls -
  IRMX Vol. 2                                                              }

  connection := dqattach (temp_token, error);
  if error <> 0 then
    begin
      gotoxy (5,12);
      write_token (output, temp_token);
      writeln (' does not exist.');
      gotoxy (5,24);
      dqexit (completion_code) { halt }
    end;
  dqdetach (connection ,error);
```

```
{    These next four lines insert an ascii nul to the end of the string so
   that Pascal-86 knows where to truncate the string for the reset and
   rewrite statements to follow. See pg. 8-17 Pascal-86 User's Guide Rev. 5 }

  input_filename[length(input_filename) + 1] := chr(0);
  output_filename[length(output_filename) + 1] := chr(0);
  error_filename[length(error_filename) + 1] := chr(0);
  constant_filename[length(constant_filename) + 1] := chr(0);
  reset (infile,input_filename);
  rewrite (outfile,output_filename);
  rewrite (errorfile,error_filename);
  rewrite (constant_file,constant_filename);
  ch := infile^;
  get (infile);
  if (ord(ch) >= 65) and (ord(ch) <= 90) then ch := char(ord(ch)+32);
  write(errorfile,ch);
  writeln(outfile,';                                                          ');
  writeln(outfile,';      p       b           e                                  ');
  writeln(outfile,';      |       |         r n     m                            ');
  writeln(outfile,';      c  2    a  b  w D d f     c                            ');
  writeln(outfile,';      n  9    d  |  | s | |     3               m I I I      ');
  writeln(outfile,';      t  1    d  o  o e o b I I 2               s A A A  A  A A');
  writeln(outfile,';      r  0    r  p  p l p a 4 3 5   AF   AR   AS w 2 1 0 IF IR IS');
  writeln(outfile,';                                                           ');
  with zero_operand do
  begin
    id := blank_token;
    id[1] := '0';
    id_type := real_symbol_type;
    index := blank_token;
    offset := 0;
    id_address := 1;
    index_address := 0;
  end;
  for i := 0 to 2 do
  begin
    write_lookahead_buffer[i].id := blank_token;
    branch_lookahead_buffer[i] := 7fffH;
    AF[i] := 0;
  end;
  for i:=0 to 1 do
  begin
    I3[i] := 0;
    mc325_buffer[i] := 0;
  end;
  expression_number := 1;
  for i := 0 to 1 do
  begin
    AIS[i] := 7fffH;
    AIR[i] := 7fffH;
```

```
end;
for i:= 0 to 2 do
begin
  IA2[i] := 0;
  AIF[i] := 7fffH;
  with write_lookahead_buffer[i] do
  begin
    id := blank_token;
    id_type := 0;
    index := blank_token;
    offset := 0;
  end;
end;
for i := 0 to max_index_register do
  index_register[i] := blank_token;
upper_letter_set := ['A'..'Z'];
delimiter :=
   [',','(',')','*','+','-','/',' ',';','{','}'
                                        ,'=','.','%',':','>','<','[',']'];
constant_program_counter := 0;
stack_pointer := dataram_address_limit;
{assign permanent constant values for 0.0, 0, 1.0, and 1}
dataram_address := 1;
str_real(0.0,token);
real_constant_value := 0.0;
insert_symbol(token,real_constant_symbol_type,symbol_value);
dataram_address := 1;
insert_symbol(false_token,boolean_constant_symbol_type,symbol_value);
dataram_address := 1;
integer_constant_value := 0;
temp_token := blank_token;
temp_token[1] := '0';
insert_symbol(temp_token,integer_constant_symbol_type,symbol_value);
temp_token := blank_token;
temp_token[1] := '0';
declare_constant(1,integer_constant_symbol_type,temp_token);
dataram_address := 3;
str_real(1.0,token);
real_constant_value := 1.0;
insert_symbol(token,real_constant_symbol_type,symbol_value);
dataram_address := 3;
insert_symbol(true_token,boolean_constant_symbol_type,symbol_value);
integer_constant_value := 1;
dataram_address := 3;
temp_token := blank_token;
temp_token[1] := '1';
insert_symbol(temp_token,integer_constant_symbol_type,symbol_value);
temp_token := blank_token;
temp_token[1] := '1';
declare_constant(3,integer_constant_symbol_type,temp_token);
program_counter := 0;
```

```
  reset_microcode_field;
  generate_Nop;
  insert_std_procedure_to_symbol_table;
  insert_standard_function_to_symbol_table;
  insert_reserved_word_to_symbol_table;
  insert_delimiter_to_symbol_table;
  create_expression(new_expression);
  expression_level := 0;
  first_expression[expression_level] := new_expression;
  no_local_variable[0] := 0;
  reset_temp_variable_address;
  for i := 0 to max_procedure_level do
    inside_function_block[i] := false;
end; { of initialize }

procedure program_heading_block;
begin
  fetch_token;  {program name}
  fetch_token;  { ( or; }
  if (token <> open_parenthesis) and (token <> semicolon) then
  begin
     writeln(errorfile);
     writeln(errorfile,'!!!! syntax error "(" or ";" expected');
     error_found;
  end;
  if (token <> semicolon) then
  begin
    if (token = open_parenthesis) then
    begin
      fetch_token; { input }
      fetch_token;
      if (token= comma) then
      begin
        fetch_token;
        fetch_token; { output }
      end;
      verify_token(token,close_parenthesis);
    end;
    fetch_token;
    verify_token(token,semicolon);
  end;
end;  { of program heading section }.
```

```
File: IO.DEF
public io;
  Procedure send_procedure;
  Procedure receive_procedure;
```

File: IO.PAS

```
module io;

$include(io.def)
$include(global.def)
$include(fetch_tk.def)
$include(arith.def)
$include(symbol_t.def)
$include(utility.def)
$include(code_gen.def)
$include(emu_lib.def)

private io;

procedure generate_send(device:token_type;operand : operand_type;
                                        order:longint);
begin
  check_F_bus(operand);
  operand_string (operand, temp_token);
  write(outfile,';',program_counter,': fetch(');
  write_token (outfile, temp_token);
  writeln (outfile, ')');
  assign_S_bus(operand);
  Assign_R_bus(operand);
  msw := order;
  microcode_address := program_counter;
  output_microcode_field;
  program_counter := program_counter + 1;
  operand_string (operand, temp_token);
  if order = 1 then
  begin
    write(outfile,';',program_counter);
    write(outfile,': send_msw(');
    write_token (outfile, device);
    write(outfile, ',');
    write_token (outfile, temp_token);
    writeln (outfile, ')')
  end
  else
  begin
    write(outfile,';',program_counter);
    write(outfile,': send_lsw(');
    write_token (outfile, device);
    write(outfile, ',');
    write_token (outfile, temp_token);
    writeln (outfile, ')')
  end;
  am2910_opcode := CJP;
  if device = host then
  begin
    branch_opcode := if_not_hrfi;
```

```
    read_opcode := load_host;
  end
  else
  if device = network then
  begin
    branch_opcode := if_not_xrfi;
    read_opcode := load_network;
  end
  else
    write_error('unknown send device :                             ',device);
  branch_address := program_counter;
  msw := order;
  assign_S_bus(operand);
  assign_R_bus(operand);
  microcode_address := program_counter;
  output_microcode_field;
  program_counter := program_counter + 1;
end; { of generate_send }

procedure generate_receive(device:token_type;operand : operand_type;
                                                          order:longint);
begin
  generate_nop;
  clear_pipeline_stage;
  operand_string(operand,temp_token);
  if order = 1 then
  begin
    write(outfile,';',program_counter,': receive_msw(');
    write_token (outfile, device);
    write (outfile,',');
    write_token (outfile, temp_token);
    writeln(outfile,')')
  end
  else
  begin
    write(outfile,';',program_counter,': receive_lsw(');
    write_token (outfile, device);
    write (outfile,',');
    write_token (outfile, temp_token);
    writeln (outfile,')')
  end;
  am2910_opcode := CJP;
  if device = host then
  begin
    branch_opcode := if_not_hdav;
    write_opcode := write_host;
  end
  else
  if device = network then
  begin
    branch_opcode := if_not_xdav;
```

```
    write_opcode := write_network;
  end
  else
    write_error('unknown receive device :                          ',device);
  branch_address := program_counter;
  msw := order;
  assign_F_bus(operand);
  microcode_address := program_counter;
  output_microcode_field;
  program_counter := program_counter + 1;
end; { of generate_receive }

Procedure send_procedure;
var
    io_operand : operand_type;
    integer_operand : operand_type;
    device : token_type;
    send_command : token_type;
    i : longint;
begin
  clear_temp_index;
  reset_temp_variable_address;
  send_command := token;
  fetch_token;
  verify_token(token,open_parenthesis);
  fetch_token;
  device := token;
  fetch_token;
  verify_token(token,comma);
  fetch_parameter(io_operand);
  find_symbol(io_operand.id,io_operand.id_type,io_operand.id_address,found);
  if not found then
    write_error('unknown id:                                  ',io_operand.id);
  io_operand.index_address := assign_index(io_operand.index);
  symbol_type := io_operand.id_type;
  if (symbol_type = real_symbol_type) or
     (symbol_type = real_constant_symbol_type) or
     (symbol_type = real_array_symbol_type) or
     (symbol_type = boolean_symbol_type) or
     (symbol_type = boolean_constant_symbol_type) or
     (symbol_type = boolean_array_symbol_type) then
  begin
    if (write_lookahead_buffer[1].id <> blank_token) then generate_nop;
    if (write_lookahead_buffer[0].id <> blank_token) then
      if (write_lookahead_buffer[0].id = io_operand.id) then
        if (write_lookahead_buffer[0].id = io_operand.id) then
          if (write_lookahead_buffer[0].index = io_operand.index) then
            if (write_lookahead_buffer[0].offset = io_operand.offset) then
              generate_nop;
    if (send_command = send_msw) or (send_command = send) then generate_send(device,io_operand,1);
    if (send_command = send_lsw) or (send_command = send) then generate_send(device,io_operand,0);
```

```
  end
  else
  if (symbol_type = integer_symbol_type) or
    (symbol_type = integer_constant_symbol_type) or
    (symbol_type = integer_array_symbol_type) then
  begin
    assign_temp_parameter(integer_operand,integer_symbol_type);
    generate_ALU_operation(integer_operand,io_operand,zero_operand,unary_round);
    delete(integer_operand.id,1,1);
    val_integer(integer_operand.id,integer_operand.id_address,i);
    clear_pipeline_stage;
    if (send_command = send_msw) or (send_command = send) then
      generate_send(device,integer_operand,1);
    if (send_command = send_lsw) or (send_command = send) then
      generate_send(device,integer_operand,0);
  end
  else
  begin
     write_error('IO not allowed                                    ',token);
  end;
  verify_token(token,close_parenthesis);
end; { of send_procedure }

Procedure receive_procedure;
var
    io_operand : operand_type;
    device : token_type;
    receive_command : token_type;
begin
  clear_temp_index;
  reset_temp_variable_address;
  receive_command := token;
  fetch_token;
  verify_token(token,open_parenthesis);
  fetch_token;
  device := token;
  fetch_token;
  verify_token(token,comma);
  fetch_parameter(io_operand);
  if io_operand.id[1] = '#' then
    write_error('simple variable is expected :                     ',device);
  find_symbol(io_operand.id,io_operand.id_type,io_operand.id_address,found);
  if not found then
    write_error('unknown id:                                       ',io_operand.id);
  io_operand.index_address := assign_index(io_operand.index);
  symbol_type := io_operand.id_type;
  if (symbol_type = real_symbol_type) or
     (symbol_type = real_constant_symbol_type) or
     (symbol_type = real_array_symbol_type) or
     (symbol_type = boolean_symbol_type) or
     (symbol_type = boolean_constant_symbol_type) or
```

```
    (symbol_type = boolean_array_symbol_type) then
  begin
    if (receive_command = receive_msw) or (receive_command = receive) then generate_receive(device,io_operand,1);
    if (receive_command = receive_lsw) or (receive_command = receive) then generate_receive(device,io_operand,0);
  end
  else
  if (symbol_type = integer_symbol_type) or
    (symbol_type = integer_constant_symbol_type) or
    (symbol_type = integer_array_symbol_type) then
  begin
    if (receive_command = receive_msw) or (receive_command = receive) then generate_receive(device,io_operand,1);
    if (receive_command = receive_lsw) or (receive_command = receive) then generate_receive(device,io_operand,0);
    if (receive_command = receive) or
       (receive_command = receive_lsw) then
          generate_alu_operation(io_operand,io_operand,zero_operand,unary_float);
  end
  else
  begin
     write_error('IO not allowed                                        ',token);
  end;
  verify_token(token,close_parenthesis);
end; { of receive_procedure }.
```

```
File: LIB.DEF
public lib;
  procedure function_trunc(var fx : operand_type; x :operand_type);
  procedure function_round(var fx : operand_type; x : operand_type);
  procedure generate_gt_ffs(function_number :integer;f,r,s : operand_type);
  procedure function_exp(fx,x : operand_type);
  procedure function_ln(fx,x : operand_type);
  procedure function_sqrt(fx,x : operand_type);
  procedure function_sin(fx,x : operand_type);
  procedure function_cos(fx,x : operand_type);
  procedure function_tan(fx,x : operand_type);
  procedure function_asin(fx,x : operand_type);
  procedure function_acos(fx,x : operand_type);
  procedure function_atan(fx,x : operand_type);
  procedure generate_reciprocal(fx,x : operand_type);
```

```
File: LIB.PAS
module lib;
$include (emu_lib.def)
$include (lib.def)
$include(global.def)
$include(exprsion.def)
$include(exprtree.def)
$include(utility.def)
$include(fetch_tk.def)
$include(symbol_t.def)
$include(code_gen.def)
$include(declare.def)

private lib;
procedure function_trunc(var fx : operand_type;x:operand_type);
begin
  write_error('only rounding mode is supported :                    ',fx.id);
end; { of function_trunc }

procedure function_round(var fx : operand_type;x:operand_type);
begin
  generate_ALU_operation(fx,x,zero_operand,unary_round);
  generate_ALU_operation(fx,fx,zero_operand,unary_float);
  fx.id_type := integer_symbol_type;
end; { of function_round }

procedure generate_gt_ffs(function_number :integer;f,r,s : operand_type);
{ This procedure produces code that is needed to lookup a function value from
  the GT-FFS/1 function board. }
var negative_one : operand_type; { is used to assign r operand to a negative number }
    w,m : longint; { are used to store the values of write_opcode and msw }
begin
  if write_lookahead_buffer[1].id <> blank_token then generate_nop;
  if write_lookahead_buffer[0].id <> blank_token then
    if (write_lookahead_buffer[0].id = r.id) and
       (write_lookahead_buffer[0].index = r.index) and
       (write_lookahead_buffer[0].offset = r.offset) then
       generate_nop
    else
    if (write_lookahead_buffer[0].id = s.id) and
       (write_lookahead_buffer[0].index = s.index) and
       (write_lookahead_buffer[0].offset = s.offset) then
       generate_nop;

  write (outfile,'; ');
  operand_string(f,temp_token);
  write_token (outfile, temp_token);
  write (outfile,' := GT_FFS(', function_number,',');
  operand_string(r, temp_token);
  write_token(outfile,temp_token); write(outfile,',');
  operand_string(s, temp_token);
```

```
write_token (outfile,temp_token); writeln(outfile,')');

assign_real_constant(negative_one,-1.0);
case function_number of
  1 : begin
        w := 4;
        m := 0;
        r := zero_operand;
      end;
  2 : begin
        w := 4;
        m := 1;
        r := zero_operand;
      end;
  3 : begin
        w := 4;
        m := 1;
        r := negative_one;
      end;
  4 : begin
        w := 5;
        m := 0;
        r := zero_operand;
      end;
  5 : begin
        w := 5;
        m := 0;
        r := negative_one;
      end;
  6 : begin
        w := 5;
        m := 1;
      end;
  7 : begin
        w := 4;
        m := 0;
        r := negative_one;
      end;
  8 : begin
        w := 7;
        m := 0;
      end;
  9 : begin
        w := 6;
        m := 1;
        r := zero_operand;
      end;
  10: begin
        w := 6;
        m := 0;
      end;
```

```
end; { of case function_number of }
find_symbol(r.id,r.id_type,r.id_address,found); if not found then
  write_error('unknown id:                                     ',r.id);
r.index_address := assign_index(r.index);
find_symbol(s.id,s.id_type,s.id_address,found); if not found then
  write_error('unknown id:                                     ',s.id);
s.index_address := assign_index(s.index);
find_symbol(f.id,f.id_type,f.id_address,found); if not found then
  write_error('unknown id:                                     ',f.id);
f.index_address := assign_index(f.index);
microcode_address := program_counter;
AR := r.id_address + r.offset;
if (r.index <> blank_token) and  (r.index[1] <> '0') and
   (r.id[1] <> '&') and (r.id[1] <> '#') then
begin
  AIR[0] := r.index_address;
  IA1 := 1;
end;
AS := s.id_address + s.offset;
if (s.index <> blank_token) and  (s.index[1] <> '0') and
   (s.id[1] <> '&') and (s.id[1] <> '#') then
begin
  AIS[0] := s.index_address;
  IA0 := 1;
end;
output_microcode_field;
program_counter := program_counter+1;
microcode_address := program_counter;
AF[0] := f.id_address + f.offset;
if (F.index <> blank_token) and  (F.index[1] <> '0') and
   (F.id[1] <> '&') and (F.id[1] <> '#') then
begin
  AIF[0] := f.index_address;
  IA2[0] := 1;
end;
msw := m;
write_opcode := w;
if function_number = 7 then
{ allows branches to be controlled by the calling procedure function_tan.
  If the branch address is not altered by the calling procedure, the code
  will simply move to the next program location }
begin
  AM2910_opcode := CJP;
  branch_opcode := if_negative;
  branch_address := program_counter+1;
end;
output_microcode_field;
program_counter := program_counter+1;
end; { of generate_gt_ffs }

procedure generate_reciprocal(fx,x : operand_type);
```

```
const iteration = 1;
var F1,F2,two : operand_type;
    i : longint;
begin
  operand_string(fx, temp_token);
  write(outfile,'; begin { ');
  write_token(outfile, temp_token);
  write(outfile,' := 1/');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
  generate_gt_ffs(4,fx,x,x); { fx := F4(x,x) }
  { Newton Raphson's iteration }
  for i := 1 to iteration do                     { fx := fx*(2-fx*x) }
  begin
    writeln(outfile,'; Newton Raphson iteration ',i);
    reset_operand(F1);
    assign_temp_parameter(F1,real_symbol_type);
    generate_ALU_operation(F1,fx,x,multiplication);  { F1 := fx * x }
    assign_real_constant(two,2.0);
    assign_temp_parameter(F2,real_symbol_type);
    generate_ALU_operation(F2,two,F1,subtraction);   { F2 := 2.0 - F1 }
    fx.id_type := real_symbol_type;
    generate_ALU_operation(fx,fx,F2,multiplication); { fx := fx * F2 }
  end; { of i := 1 to iteration }
  operand_string(fx, temp_token);
  write(outfile,'; end { ');
  write_token(outfile, temp_token);
  write(outfile,' := 1/');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
end; { of generate_reciprocal }

procedure function_exp(fx,x : operand_type);
var k,invln2,ln2,r : operand_type;
    z,w,p1,p2,p3,half,two : operand_type;
begin
  operand_string(fx, temp_token);
  write(outfile,'; begin { ');
  write_token(outfile, temp_token);
  write(outfile,' := exp(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
  writeln(outfile,';    { calculate k := round(x/ln(2)) }');
  assign_temp_parameter(k,real_symbol_type);
  assign_real_constant(invln2,1.0/ln(2.0));
  generate_ALU_operation(k,x,invln2,multiplication); { k := x / ln(2) }
  generate_ALU_operation(k,k,zero_operand,unary_round); { k := round(k) }
  writeln(outfile,';    { calculate the remainder r := x - float(k)*ln2 }');
```

```
  assign_real_constant(ln2,ln(2));
  assign_temp_parameter(r,real_symbol_type);
  generate_ALU_operation(r,k,zero_operand,unary_float); { r := float(k) }
  generate_ALU_operation(r,r,ln2,multiplication); { r := r*ln2 }
  generate_ALU_operation(r,x,r,subtraction); { r := x - r }
  { assign p1,p2,p3 }
  assign_real_constant(p1,4.9987178778e-2);
  assign_real_constant(p2,4.1602886268e-3);
  assign_real_constant(p3,2.4999999950e-1);
  writeln(outfile,';    { calculate z := r*r }');
  assign_temp_parameter(z,real_symbol_type);
  generate_ALU_operation(z,r,r,multiplication); { z := r*r }
  writeln(outfile,';    { calculate w := r*(z*p2+p3) }');
  assign_temp_parameter(w,real_symbol_type);
  generate_ALU_operation(w,z,p2,multiplication); { w := z*p2 }
  generate_ALU_operation(w,w,p3,addition); { w := w+p3 }
  generate_ALU_operation(w,r,w,multiplication); { w := r*w }
  writeln(outfile,';    { calculate fx(r) = 0.5 + w/(0.5+z*p1-w) }');
  assign_real_constant(half,0.5);
  generate_ALU_operation(fx,z,p1,multiplication); { fx := z*p1 }
  generate_ALU_operation(fx,half,fx,addition); { fx := 0.5+fx }
  generate_ALU_operation(fx,fx,w,subtraction); { fx := fx-w }
  generate_reciprocal(z,fx); { z := 1/fx , z is used as temporary variable }
  generate_ALU_operation(fx,w,z,multiplication); { fx := w*z }
  generate_ALU_operation(fx,half,fx,addition); { fx := 0.5 + fx }
  writeln(outfile,';    { calculate fx := F1(k)*fx(r)*2 }');
  generate_gt_ffs(1,k,k,k); { k := F1(k,k) }
  generate_ALU_operation(fx,k,fx,multiplication); { fx := k*fx }
  assign_real_constant(two,2.0);
  generate_ALU_operation(fx,fx,two,multiplication); { fx := fx*2 }
  operand_string(fx, temp_token);
  write(outfile,'; end { ');
  write_token(outfile, temp_token);
  write(outfile,' := exp(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
end; { of function_exp }

Procedure function_ln(fx,x : operand_type);
var one,ln2,r,p,a1,a2,a3,a4,a5,a6,a7,a8 : operand_type;
begin
  operand_string(fx, temp_token);
  write(outfile,'; begin { ');
  write_token(outfile, temp_token);
  write(outfile,' := ln(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
  writeln(outfile,';    { calculate fx := F2(x)*ln2 & r := F3(x)-1 } ');
  assign_real_constant(ln2,ln(2));
```

```
  assign_real_constant(one,1.0);
  assign_temp_parameter(r,real_symbol_type);
  generate_gt_ffs(2,fx,x,x); { fx := F2(x) }
(*  generate_ALU_operation(fx,fx,zero_operand,unary_float); temporary fixes for the EPROM.
  The EPROM should have returned a real number. *)
  generate_gt_ffs(3,r,x,x);  { r := F3(x) }
  generate_ALU_operation(fx,fx,ln2,multiplication); { fx := fx*ln2 }
  generate_ALU_operation(r,r,one,subtraction); { r := r-1 }
  writeln(outfile,';    { calculate p := r(a1+r(a2+r(a3+r(a4+r(a5+r(a6+r(a7+r(a8)))))))) }');
  assign_real_constant(a1, 0.9999964239);
  assign_real_constant(a2,-0.4998741238);
  assign_real_constant(a3, 0.3317990358);
  assign_real_constant(a4,-0.2407338084);
  assign_real_constant(a5, 0.1676540711);
  assign_real_constant(a6,-0.0953293897);
  assign_real_constant(a7, 0.0360884937);
  assign_real_constant(a8,-0.0064535442);
  assign_temp_parameter(p,real_symbol_type);
  generate_ALU_operation(p,a8,r,multiplication); { p := a8*r }
  generate_ALU_operation(p,a7,p,addition) ; { p := a7+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a6,p,addition) ; { p := a6+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a5,p,addition) ; { p := a5+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a4,p,addition) ; { p := a4+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a3,p,addition) ; { p := a3+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a2,p,addition) ; { p := a2+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a1,p,addition) ; { p := a1+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  writeln(outfile,';    { calculate fx := fx + p }');
  generate_ALU_operation(fx,fx,p,addition);
  operand_string(fx, temp_token);
  write(outfile,'; end { ');
  write_token(outfile, temp_token);
  write(outfile,' := ln(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
end; { of function_ln }

procedure function_sqrt(fx,x : operand_type);
const iteration = 1;
var F1,half,three : operand_type;
    i : longint;
begin
  operand_string(fx, temp_token);
  write(outfile,'; begin { ');
```

```
  write_token(outfile, temp_token);
  write(outfile,' := sqrt(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
  generate_gt_ffs(5,fx,x,x); { fx := F5(x,x) }
  assign_temp_parameter(F1,real_symbol_type);
  assign_real_constant(half,0.5);
  assign_real_constant(three,3.0);
  { Newton Raphson's iteration }
  for i := 1 to iteration do                    { fx := 0.5*fx*(3-fx*fx*x) }
  begin
    writeln(outfile,'; Newton Raphson iteration ',i);
    generate_ALU_operation(F1,fx,fx,multiplication);  { F1 := fx * fx }
    generate_ALU_operation(F1,x,F1,multiplication); { F1 := x * F1 }
    generate_ALU_operation(F1,three,F1,subtraction); { F1 := 3 - F1 }
    generate_ALU_operation(F1,fx,F1,multiplication); { F1 := fx*F1 }
    generate_ALU_operation(fx,half,F1,multiplication); { fx := 0.5*F1 }
  end; { of i := 1 to iteration }
  generate_ALU_operation(fx,x,fx,multiplication); { fx := fx*x }
  operand_string(fx, temp_token);
  write(outfile,'; end { ');
  write_token(outfile, temp_token);
  write(outfile,' := sqrt(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
end; { of function_sqrt }

Procedure function_sin(fx,x : operand_type);
var one,invpi,pi1,k,r,z,p,a1,a2,a3,a4 : operand_type;
begin
  operand_string(fx, temp_token);
  write(outfile,'; begin { ');
  write_token(outfile, temp_token);
  write(outfile,' := sin(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
  writeln(outfile,';   { calculate k := round(x/pi) }');
  assign_temp_parameter(k,real_symbol_type);
  assign_real_constant(invpi,1/pi);
  assign_real_constant(pi1,pi);
  generate_ALU_operation(k,x,invpi,multiplication);      { k := x/pi }
  generate_ALU_operation(k,k,zero_operand,unary_round); { k := round(k) }
  writeln(outfile,';   { calculate r := (x - float(k)*pi) & z := r*r }');
  assign_temp_parameter(r,real_symbol_type);
  assign_temp_parameter(z,real_symbol_type);
  generate_ALU_operation(r,k,zero_operand,unary_float); { r := float(k) }
  generate_ALU_operation(r,r,pi1,multiplication); { r := r*pi }
  generate_ALU_operation(r,x,r,subtraction); { r : = x-r }
```

```
  generate_ALU_operation(z,r,r,multiplication); { z := r*r }
  writeln(outfile,';    { calculate p := r(1+z(a1+z(a2+z(a3+z(a4))))) }');
  assign_temp_parameter(p,real_symbol_type);
  assign_real_constant(a1,-1.666665668e-1);
  assign_real_constant(a2, 8.333025139e-3);
  assign_real_constant(a3,-1.980741872e-4);
  assign_real_constant(a4, 2.601903036e-6);
  assign_real_constant(one,1.0);
  generate_ALU_operation(p,a4,z,multiplication); { p := a4*z }
  generate_ALU_operation(p,a3,p,addition) ; { p := a3+p }
  generate_ALU_operation(p,z,p,multiplication); { p := z*p }
  generate_ALU_operation(p,a2,p,addition) ; { p := a2+p }
  generate_ALU_operation(p,z,p,multiplication); { p := z*p }
  generate_ALU_operation(p,a1,p,addition) ; { p := a1+p }
  generate_ALU_operation(p,z,p,multiplication); { p := z*p }
  generate_ALU_operation(p,one,p,addition) ; { p := 1+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_GT_FFS(6,fx,k,p);  { fx := F6(k,p); }
  operand_string(fx, temp_token);
  write(outfile,'; end { ');
  write_token(outfile, temp_token);
  write(outfile,' := sin(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
end; { of function_sin }

Procedure function_cos(fx,x : operand_type);
var one,invpi,piov2,pi1,k,r,z,x1,p,a1,a2,a3,a4 : operand_type;
begin
  operand_string(fx, temp_token);
  write(outfile,'; begitn { ');
  write_token(outfile, temp_token);
  write(outfile,' := cos(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
  writeln(outfile,';    { calculate x1 := x+pi/2 }');
  assign_temp_parameter(x1,real_symbol_type);
  assign_real_constant(invpi,1/pi);
  assign_real_constant(piov2,pi/2);
  generate_ALU_operation(x1,x,piov2,addition);
  writeln(outfile,';    { calculate k := round(x1/pi) }');
  assign_temp_parameter(k,real_symbol_type);
  assign_real_constant(pi1,pi);
  generate_ALU_operation(k,x1,invpi,multiplication);       { k := x1/pi }
  generate_ALU_operation(k,k,zero_operand,unary_round); { k := round(k) }
  writeln(outfile,';    { calculate r := (x1 - float(k)*pi) }');
  assign_temp_parameter(r,real_symbol_type);
  assign_temp_parameter(z,real_symbol_type);
  generate_ALU_operation(r,k,zero_operand,unary_float); { r := float(k) }
```

```
  generate_ALU_operation(r,r,pil,multiplication); { r := r*pi }
  generate_ALU_operation(r,x1,r,subtraction); { r : = x1-r }
  generate_ALU_operation(z,r,r,multiplication); { z := r*r }
  writeln(outfile,';   { calculate p := r(1+z(a1+z(a2+z(a3+z(a4))))) }');
  assign_temp_parameter(p,real_symbol_type);
  assign_real_constant(a1,-1.666665668e-1);
  assign_real_constant(a2, 8.333025139e-3);
  assign_real_constant(a3,-1.980741872e-4);
  assign_real_constant(a4, 2.601903036e-6);
  assign_real_constant(one,1.0);
  generate_ALU_operation(p,a4,z,multiplication); { p := a4*z }
  generate_ALU_operation(p,a3,p,addition) ; { p := a3+p }
  generate_ALU_operation(p,z,p,multiplication); { p := z*p }
  generate_ALU_operation(p,a2,p,addition) ; { p := a2+p }
  generate_ALU_operation(p,z,p,multiplication); { p := z*p }
  generate_ALU_operation(p,a1,p,addition) ; { p := a1+p }
  generate_ALU_operation(p,z,p,multiplication); { p := z*p }
  generate_ALU_operation(p,one,p,addition) ; { p := 1+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_GT_FFS(6,fx,k,p);  { fx := F6(k,p); }
  operand_string(fx, temp_token);
  write(outfile,'; end { ');
  write_token(outfile, temp_token);
  write(outfile,' := cos(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
end; { of function_cos }

Procedure function_tan(fx,x : operand_type);
var one,invpiov2,piov2,k,t,r,z,x1,p,q,p1,q2,q1 : operand_type;
    povq_address : longint; { starting address for the evaluation of p/q }
    qovp_address : longint; { starting address for the evaluation of q/p }
begin
  operand_string(fx, temp_token);
  write(outfile,'; begin { ');
  write_token(outfile, temp_token);
  write(outfile,' := tan(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
  writeln(outfile,';   { calculate k := round(x*4/pi) }');
  assign_temp_parameter(k,real_symbol_type);
  assign_real_constant(piov2,pi/2);
  assign_real_constant(invpiov2,2/pi);
  generate_ALU_operation(k,x,invpiov2,multiplication);  { k := x*4/pi }
  generate_ALU_operation(k,k,zero_operand,unary_round); { k := round(k) }
  writeln(outfile,';   { calculate r := (x - float(k)*pi/4) }');
  assign_temp_parameter(r,real_symbol_type);
  generate_ALU_operation(r,k,zero_operand,unary_float); { r := float(k) }
  generate_ALU_operation(r,r,piov2,multiplication); { r := r*pi/4 }
```

```
  generate_ALU_operation(r,x,r,subtraction); { r : = x-r }
  writeln(outfile,';    { calculate z := r*r }');
  assign_temp_parameter(z,real_symbol_type);
  generate_ALU_operation(z,r,r,multiplication); { z := r*r }
  writeln(outfile,';    : calculate q := (1.0 + (q2*z + q1)*z)');
  assign_real_constant(q2,9.71685835e-3);
  assign_real_constant(q1,-4.29135777e-1);
  assign_real_constant(one,1.0);
  assign_temp_parameter(q,real_symbol_type);
  generate_ALU_operation(q,q2,z,multiplication); { q := q2*z }
  generate_ALU_operation(q,q,q1,addition); { q := q + q1 }
  generate_ALU_operation(q,q,z,multiplication); { q := q*z }
  generate_ALU_operation(q,one,q,addition); { q := 1.0 + q }
  writeln(outfile,';    : calculate p := (p1*z*r + r)');
  assign_temp_parameter(p,real_symbol_type);
  assign_real_constant(p1,-9.58017723e-2);
  generate_ALU_operation(p,p1,z,multiplication);  { p := p1*z }
  generate_ALU_operation(p,p,r,multiplication);  { p := p*r }
  generate_ALU_operation(p,p,r,addition); { p := p + r }
  writeln(outfile,';    { if k is even then fx := p/q else fx := q/p }');
  assign_temp_parameter(t,real_symbol_type);
  generate_gt_ffs(7,t,k,k);  { t := F7(k) }
  writeln(outfile,';    { calculate fx := p/q }');
  povq_address := program_counter;
  generate_reciprocal(fx,q);  { fx := 1/q }
  generate_ALU_operation(fx,p,fx,multiplication); { fx := p*fx }
  generate_Nop;
  microcode_address := program_counter;
  am2910_opcode := CJP;
  branch_opcode := unconditional;
  writeln(outfile,';30: unconditional branch ');
  output_microcode_field;
  program_counter := program_counter + 1;
  writeln(outfile,';    { calculate fx := q/p }');
  writeln(outfile,'b ',povq_address-1,' ',program_counter);
  qovp_address := program_counter;
  generate_reciprocal(fx,p);  { fx := 1/p }
  generate_ALU_operation(fx,q,fx,multiplication); { fx := q*fx }
  generate_Nop;
  generate_Nop;
  writeln(outfile,'b ',qovp_address-1,' ',program_counter);
  writeln(outfile,';    fx := F8(LF7,x,fx)');
  generate_gt_ffs(6,fx,k,fx);
  operand_string(fx, temp_token);
  write(outfile,'; end { ');
  write_token(outfile, temp_token);
  write(outfile,' := tan(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
end; { of function_tan }
```

```
Procedure function_asin(fx,x : operand_type);
var piov2,one,t,r,p,sqrt1_r,a0,a1,a2,a3,a4,a5,a6,a7 : operand_type;
begin
  operand_string(fx, temp_token);
  write(outfile,'; begin { ');
  write_token(outfile, temp_token);
  write(outfile,' := asin(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
  writeln(outfile,';  r := |x| ');
  assign_temp_parameter(r,real_symbol_type);
  generate_gt_ffs(9,r,zero_operand,x); { r := abs(x) if -1 <= x <= 1 else r = NAN }
  writeln(outfile,';    { calculate p := pi/2 - sqrt(1-r)(a0+r(a1+r(a2+r(a3+r(a4+r(a5+r(a6+r(a7))))))) }');
  assign_temp_parameter(p,real_symbol_type);
  assign_temp_parameter(sqrt1_r,real_symbol_type);
  assign_temp_parameter(t,real_symbol_type);
  assign_real_constant(a0, 1.5707963050);
  assign_real_constant(a1,-0.2145988016);
  assign_real_constant(a2, 0.0889789874);
  assign_real_constant(a3,-0.0501743046);
  assign_real_constant(a4, 0.0308918810);
  assign_real_constant(a5,-0.0170881256);
  assign_real_constant(a6, 0.0066700901);
  assign_real_constant(a7,-0.0012624911);
  assign_real_constant(piov2,pi/2);
  assign_real_constant(one,1.0);
  generate_ALU_operation(t,one,r,subtraction);
  function_sqrt(sqrt1_r,t);
  generate_ALU_operation(p,r,a7,multiplication) ; { p := r*a7 }
  generate_ALU_operation(p,a6,p,addition) ; { r := a6+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a5,p,addition) ; { r := a5+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a4,p,addition) ; { r := a4+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a3,p,addition) ; { r := a3+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a2,p,addition) ; { r := a2+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a1,p,addition) ; { r := a1+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a0,p,addition) ; { r := a0+p }
  generate_ALU_operation(p,sqrt1_r,p,multiplication); { p := sqrt1_r*p }
  generate_ALU_operation(p,piov2,p,subtraction); { p := pi/2 - p }
  writeln(outfile,';    { fx := (1^sign(x))*p }');
  generate_GT_FFS(10,fx,x,p);  { fx := F10(x,p); }
  operand_string(fx, temp_token);
  write(outfile,'; end { ');
  write_token(outfile, temp_token);
```

```
  write(outfile,' := asin(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
end; { of function_asin }

Procedure function_acos(fx,x : operand_type);
var piov2,one,t,r,p,sqrtl_r,a0,a1,a2,a3,a4,a5,a6,a7 : operand_type;
begin
  operand_string(fx, temp_token);
  write(outfile,'; begin { ');
  write_token(outfile, temp_token);
  write(outfile,' := acos(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
  writeln(outfile,';  r := |x| ');
  assign_temp_parameter(r,real_symbol_type);
  generate_gt_ffs(9,r,zero_operand,x); { r := abs(x) if -1 <= x <= 1 else r = NAN }
  writeln(outfile,';    { calculate p := pi/2 - sqrt(1-r)(a0+r(a1+r(a2+r(a3+r(a4+r(a5+r(a6+r(a7))))))) }');
  assign_temp_parameter(p,real_symbol_type);
  assign_temp_parameter(sqrtl_r,real_symbol_type);
  assign_temp_parameter(t,real_symbol_type);
  assign_real_constant(a0, 1.5707963050);
  assign_real_constant(a1,-0.2145988016);
  assign_real_constant(a2, 0.0889789874);
  assign_real_constant(a3,-0.0501743046);
  assign_real_constant(a4, 0.0308918810);
  assign_real_constant(a5,-0.0170881256);
  assign_real_constant(a6, 0.0066700901);
  assign_real_constant(a7,-0.0012624911);
  assign_real_constant(piov2,pi/2);
  assign_real_constant(one,1.0);
  generate_ALU_operation(t,one,r,subtraction);
  function_sqrt(sqrtl_r,t);
  generate_ALU_operation(p,r,a7,multiplication) ; { p := r*a7 }
  generate_ALU_operation(p,a6,p,addition) ; { r := a6+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a5,p,addition) ; { r := a5+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a4,p,addition) ; { r := a4+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a3,p,addition) ; { r := a3+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a2,p,addition) ; { r := a2+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a1,p,addition) ; { r := a1+p }
  generate_ALU_operation(p,r,p,multiplication); { p := r*p }
  generate_ALU_operation(p,a0,p,addition) ; { r := a0+p }
  generate_ALU_operation(p,sqrtl_r,p,multiplication); { p := sqrtl_r*p }
  generate_ALU_operation(p,piov2,p,subtraction); { p := pi/2 - p }
```

```
  writeln(outfile,';    { fx := (1^sign(x))*p }');
  generate_GT_FFS(10,fx,x,p);  { fx := F10(x,p); }
  generate_ALU_operation(fx,piov2,fx,subtraction);
  operand_string(fx, temp_token);
  write(outfile,'; end { ');
  write_token(outfile, temp_token);
  write(outfile,' := acos(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
end; { of function_acos }

Procedure function_atan(fx,x : operand_type);
var one,t,r,p,z,a1,a2,a3,a4,a5,a6,a7,a8,piov2,mpiov2 : operand_type;
    start_address,address1,address2 : longint;
begin
  operand_string(fx, temp_token);
  write(outfile,'; begin { ');
  write_token(outfile, temp_token);
  write(outfile,' := atan(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') }');
  writeln(outfile,';    { r = x if (-1 <= x <= 1) else r = 1/x }');
  assign_real_constant(one,1.0);
  assign_temp_parameter(r,real_symbol_type);
  assign_temp_parameter(t,real_symbol_type);
  branch_lookahead_buffer[2] := if_negative;
  generate_ALU_operation(t,one,x,subtraction); { if x > 1 then branch to evaluate 1/x }
  start_address := program_counter-1;
  branch_lookahead_buffer[2] := if_negative;
  generate_ALU_operation(t,x,one,addition); { if x < -1 then branch to evaluate 1/x }
  address1 := program_counter-1;
  branch_lookahead_buffer[2] := unconditional;
  generate_ALU_operation(r,x,zero_operand,addition); { r := x }
  address2 := program_counter-1;
  clear_pipeline_stage;
  writeln(outfile,'b ',start_address+2,' ',program_counter);
  writeln(outfile,'b ',address1+2,' ',program_counter);
  writeln(outfile,';    { calculate r := 1/x }');
  generate_reciprocal(r,x); { r := 1/x }
  clear_pipeline_stage;
  writeln(outfile,'b ',address2+2,' ',program_counter);
  writeln(outfile,';    { calculate fx := r(a1+z(a2+z(a3+z(a4+z(a5+z(a6+z(a7+z(a8))))))) }');
  assign_temp_parameter(z,real_symbol_type);
  assign_temp_parameter(p,real_symbol_type);
  assign_real_constant(a1, 0.9999993329);
  assign_real_constant(a2,-0.3332985605);
  assign_real_constant(a3, 0.1994653599);
  assign_real_constant(a4,-0.1390853351);
  assign_real_constant(a5, 0.0964200441);
```

```
assign_real_constant(a6,-0.0559098861);
assign_real_constant(a7, 0.0218612288);
assign_real_constant(a8,-0.0040540580);
generate_ALU_operation(z,r,r,multiplication); { z := r*r }
generate_ALU_operation(p,z,a8,multiplication) ; { p := z*a8 }
generate_ALU_operation(p,a7,p,addition) ; { p := a7+p }
generate_ALU_operation(p,z,p,multiplication); { p := z*p }
generate_ALU_operation(p,a6,p,addition) ; { r := a6+p }
generate_ALU_operation(p,z,p,multiplication); { p := z*p }
generate_ALU_operation(p,a5,p,addition) ; { r := a5+p }
generate_ALU_operation(p,z,p,multiplication); { p := z*p }
generate_ALU_operation(p,a4,p,addition) ; { r := a4+p }
generate_ALU_operation(p,z,p,multiplication); { p := z*p }
generate_ALU_operation(p,a3,p,addition) ; { r := a3+p }
generate_ALU_operation(p,z,p,multiplication); { p := z*p }
generate_ALU_operation(p,a2,p,addition) ; { r := a2+p }
generate_ALU_operation(p,z,p,multiplication); { p := z*p }
generate_ALU_operation(p,a1,p,addition) ; { r := a1+p }
generate_ALU_operation(fx,r,p,multiplication); { fx := r*p }
start_address := program_counter;
assign_real_constant(piov2,pi/2);
branch_lookahead_buffer[2] := if_negative;
generate_ALU_operation(t,x,one,addition); { if x < -1 then branch
                                             to evaluate fx := - pi/2 - fx }
branch_lookahead_buffer[2] := if_negative;
generate_ALU_operation(t,one,x,subtraction); { if x > 1 then branch
                                             to evaluate fx := pi/2 - fx }
generate_nop;
generate_nop;
microcode_address := program_counter;
AM2910_opcode := CJP;
branch_opcode := unconditional;
writeln(outfile,';',program_counter,': unconditional branch');
output_microcode_field;
program_counter := program_counter+1;
clear_pipeline_stage;
address1 := program_counter;
writeln(outfile,'b ',start_address+2,' ',program_counter);
writeln(outfile,';    { calculate fx := - pi/2 - fx }');
branch_lookahead_buffer[2] := unconditional;
assign_real_constant(mpiov2,-1.5707963943);
generate_ALU_operation(fx,mpiov2,fx,subtraction); { fx := - pi/2 - fx }
clear_pipeline_stage;
writeln(outfile,'b ',start_address+3,' ',program_counter);
writeln(outfile,';    { calculate fx := pi/2 - fx }');
generate_ALU_operation(fx,piov2,fx,subtraction); { fx := pi/2 - fx}
clear_pipeline_stage;
writeln(outfile,'b ',start_address+4,' ',program_counter);
writeln(outfile,'b ',address1+2,' ',program_counter);
operand_string(fx, temp_token);
write(outfile,'; end { ');
```

```
  write_token(outfile, temp_token);
  write(outfile,' := atan(');
  operand_string(x, temp_token);
  write_token(outfile,temp_token);
  writeln(outfile,') )');
end; { of function_atan }.
```

```
File: MAINBODY.DEF
public mainbody;
  Procedure while_statement;
  Procedure if_statement;
  Procedure for_statement;
  Procedure compound_statement;
  Procedure program_main_block;
  Procedure null_statement;
```

```
File: MAINBODY.PAS
module mainbody;

$include(global.def)
$include(utility.def)
$include(init.def)
$include(fetch_tk.def)
$include(symbol_t.def)
$include(code_gen.def)
$include(exprsion.def)
$include(exprtree.def)
$include(declare.def)
$include(io.def)
$include(arith.def)
$include(stdprocd.def)
$include(mainbody.def)

private mainbody;

$include(if_while.pas)
$include(for_stat.pas)
$include(procedur.pas)

Procedure null_statement;
begin
end; { of null_statement }

procedure compound_statement;
begin
  fetch_token;
  find_symbol(token,symbol_type,symbol_value,found);
  if not found then
  begin
    writeln(errorfile);
    writeln(errorfile,'!!! error, unknown id: "',token,'"');
    error_found;
  end;
  if token = begin_token then
  begin
    compound_statement;
    while token[1]=';' do compound_statement;
    verify_token(token,end_token);
    fetch_token;
  end
  else
  if token = while_token then
    while_statement
  else
  if token = repeat_token then
    repeat_statement
  else
```

```
  if token = if_token then
    if_statement
  else
  if token = for_token then
    for_statement
  else
  if (symbol_type = procedure_symbol_type) then
    procedure_call
  else
  if (symbol_type = standard_procedure_symbol_type) then
     standard_procedure_block
  else
  if (symbol_type = real_symbol_type) or
     (symbol_type = integer_symbol_type) or
     (symbol_type = boolean_symbol_type) then
  begin
    if symbol_type = real_symbol_type then
      constant_assignment_type := real_constant_symbol_type
    else
      constant_assignment_type := integer_constant_symbol_type;
    assignment_statement;
  end
  else
  if (symbol_type = real_array_symbol_type) then
  begin
    constant_assignment_type := real_array_symbol_type;
    index_assignment_statement;
  end
  else
  if (symbol_type = integer_array_symbol_type) then
  begin
    constant_assignment_type := integer_array_symbol_type;
    index_assignment_statement;
  end
  else
  if (symbol_type = boolean_array_symbol_type) then
  begin
    constant_assignment_type := boolean_array_symbol_type;
    index_assignment_statement;
  end;
end; { of compound_statement }

Procedure program_main_block;
var start_address : longint;
begin
  start_address := program_counter;
  branch_lookahead_buffer[0] := unconditional;
  generate_nop;
  while  ( (token = var_declaration)   or (token = const_declaration)
           or (token = procedure_heading) or (token=function_heading) ) do
  begin
```

```
    if (token=var_declaration) then
      var_declaration_block
    else
    if (token=const_declaration) then
      const_declaration_block
    else
    if (token=type_declaration) then
      type_declaration_block
    else
    if (token=procedure_heading) then
      procedure_main_block
    else
    if (token=function_heading) then
      function_main_block;
  end;
  if (token=begin_token) then
  begin
    writeln(outfile,'b ',start_address,' ',program_counter);
    compound_statement;
    while token[1]=';' do compound_statement;
  end
  else
  begin
     writeln(errorfile);
     writeln(errorfile,'!!!! syntax errror, begin expected');
     error_found;
  end;
  verify_token(token,end_token);
  fetch_token;
end; { of program_main_block }.
```

```
File: MAKEFILE
PASFLAGS = large optimize(1) symbolspace(64) debug
PLMFLAGS = large optimize(3) debug


SOURCES = \
          arith.pas \
          bit_func.plm \
          code_gen.pas \
          declare.pas \
          emu_lib.pas \
          exprsion.pas \
          exprtree.pas \
          fetch_tk.pas \
          global.pas \
          hex_conv.pas \
          ieee_cnv.pas \
          init.pas \
          io.pas \
          lib.pas \
          mainbody.pas \
          stdprocd.pas \
          symbol_t.pas \
          utility.pas

OBJECTS = \
          arith.obj \
          bit_func.obj \
          code_gen.obj \
          declare.obj \
          emu_lib.obj \
          exprsion.obj \
          exprtree.obj \
          fetch_tk.obj \
          global.obj \
          hex_conv.obj \
          ieee_cnv.obj \
          init.obj \
          io.obj \
          lib.obj \
          mainbody.obj \
          stdprocd.obj \
          symbol_t.obj \
          utility.obj


compiler: compiler.obj compiler.lib
          submit :PFP:csd/PASbndl( compiler, 'compiler.obj,compiler.lib', debug )


compiler.obj:       compiler.PAS
```

```
        pas286 compiler.PAS $(PASFLAGS)


compiler.lib:       $(OBJECTS)


.PAS.obj:
        pas286 $< $(PASFLAGS)
        submit :PFP:csd/lib( compiler.lib, $* )


.PLM.obj:
        plm286 $< $(PLMFLAGS)
        submit :PFP:csd/lib( compiler.lib, $* )


clean:
        delete compiler,*.lst,*.obj,*.mp?,*.lib
```

```
File: PLM_LIB.DEF
public plm_help;
   procedure plm_halt;
```

```
File: PQCLOSE.DEF
public pqclose;
   procedure pqclose(var f : text);
```

File: PROCEDUR.PAS

```
procedure parameter_list;
var i,j : longint;
    index_number : longint; { use to point to index register for call by value parameter }
    var_type : longint;  { use to store the type of the parameter }
    temp_dataram_address : longint; { use to store temporarily the dataram address for call by value parameter }
    parameter_type : longint;
begin
  index_number := 1;
  if token = open_parenthesis then
  begin
    repeat
      i := 0;
      fetch_token; { variable name / var }
      if token = var_declaration then
      begin
        parameter_type := call_by_reference;
        fetch_token; { variable name }
      end
      else
        parameter_type := call_by_value;
      i := i+1;
      symbol_array[i] := token; { insert token to symbol array }
      fetch_token; { , }
      while token = comma do
      begin
        fetch_token; { variable_name };
        i := i+1;
        symbol_array[i] := token; { insert token to symbol array }
        fetch_token;
      end;
      verify_token(token,colon);
      fetch_token; { variable_type }
      if (token = real_token) then
      begin
        var_type := real_symbol_type;
      end
      else
      if (token = integer_token) then
      begin
        var_type := integer_symbol_type;
      end
      else
      if (token = boolean_token) then
      begin
        var_type := boolean_symbol_type;
      end
      else
      begin
        writeln(errorfile);
        writeln(errorfile,'!!! error, unsupported parameter type : "',token,'"');
```

```
        error_found;
      end;
      symbol_type := var_type;
      for j := 1 to i do    { insert symbol_array to symbol table }
      begin
        new(new_parameter);
        new_parameter^.id := symbol_array[j];
        new_parameter^.id_type := symbol_type;
        new_parameter^.parameter_type := parameter_type;
        new_parameter^.next := nil;
        if parameter_type = call_by_value then
        begin
          insert_symbol(symbol_array[j],symbol_type,symbol_value);
          new_parameter^.address := symbol_value;
        end
        else { call_by_reference }
        begin
          insert_symbol(symbol_array[j],symbol_type,symbol_value);
          new_parameter^.address := symbol_value;
        end;
        if procedure_link^.parameter_link = nil then
        begin
          procedure_link^.parameter_link := new_parameter;
        end
        else
        begin
          current_parameter := procedure_link^.parameter_link;
          while current_parameter^.next <> nil do current_parameter := current_parameter^.next;
          current_parameter^.next := new_parameter;
        end;
        no_local_variable[procedure_level] := no_local_variable[procedure_level]+1;
        local_variable[procedure_level,no_local_variable[procedure_level]] := symbol_array[j];
      end;
      fetch_token;
    until ( token <> semicolon);
    verify_token(token,close_parenthesis);
    fetch_token;
  end;
end; { of parameter_list }

Procedure procedure_main_block;
var i : longint;
  procedure_name : token_type;
begin
  for i := 0 to max_index_register do index_register[i] := blank_token;
  fetch_token; { procedure_name }
  procedure_name := token;
  writeln(outfile,'; { procedure ',procedure_name);
  symbol_type := procedure_symbol_type;
  symbol_value := program_counter;
  insert_symbol(procedure_name,symbol_type,symbol_value);
```

```
  no_local_variable[procedure_level] := no_local_variable[procedure_level]+1;
  local_variable[procedure_level,no_local_variable[procedure_level]] := procedure_name;
  procedure_level := procedure_level+1;
  if procedure_level > max_procedure_level then
  begin
    writeln(errorfile);
    writeln(errorfile,'!!! error, maximum procedure nesting level exceeded, max = ',max_procedure_level);
    error_found;
  end;
  no_local_variable[procedure_level] := 0;
  fetch_token;
  parameter_list;
{ find_symbol(procedure_name,symbol_type,symbol_value,found);
  current_parameter := procedure_link^.parameter_link;
  while current_parameter <> nil do
  begin
    writeln(outfile,'----out of parameter list-----');
    writeln(outfile,'current_parameter id      : ',current_parameter^.id);
    writeln(outfile,'current_parameter id_type : ',current_parameter^.id_type);
    writeln(outfile,'current_parameter address : ',current_parameter^.address);
    current_parameter := current_parameter^.next;
  end; }
  verify_token(token,semicolon);
  fetch_token;
  program_main_block;
  verify_token(token,semicolon);
  if no_local_variable[procedure_level] > max_local_variable then
    write_error('maximum number of local variable exceeded limit   ',token);
  for i := 1 to no_local_variable[procedure_level] do
  begin
{   writeln(outfile,'delete from symbol table : "',local_variable[procedure_level,i],'"'); }
    delete_symbol(local_variable[procedure_level,i]);
  end;
  if write_lookahead_buffer[1].id <> blank_token then generate_Nop;
  writeln(outfile,';',program_counter,': return');
  microcode_address := program_counter;
  AM2910_opcode := CRTN;
  branch_opcode := unconditional;
  output_microcode_field;
  program_counter := program_counter + 1;
  fetch_token;
  for i := 0 to max_index_register do index_register[i] := blank_token;
  procedure_level := procedure_level-1;
  writeln(outfile,'; end of procedure ',procedure_name,' }' );
end; { of procedure_main_block }

Procedure function_main_block;
var i : longint;
  function_name : token_type;
begin
  for i := 0 to max_index_register do index_register[i] := blank_token;
```

```
fetch_token; { function_name }
function_name := token;
writeln(outfile,'; { function ',function_name);
symbol_type := function_symbol_type;
symbol_value := program_counter;
insert_symbol(function_name,symbol_type,symbol_value);
no_local_variable[procedure_level] := no_local_variable[procedure_level]+1;
local_variable[procedure_level,no_local_variable[procedure_level]] := function_name;
procedure_level := procedure_level+1;
inside_function_block[procedure_level] := true;
if procedure_level > max_procedure_level then
begin
  writeln(errorfile);
  writeln(errorfile,'!!! error, maximum procedure nesting level exceeded, max = ',max_procedure_level);
  error_found;
end;
no_local_variable[procedure_level] := 0;
fetch_token;
parameter_list;
find_symbol(function_name,symbol_type,symbol_value,found);
verify_token(token,colon);
fetch_token;
if token = real_token then
  symbol_type := real_symbol_type
else
if token = integer_token then
  symbol_type := integer_symbol_type
else
if token = boolean_token then
  symbol_type := boolean_symbol_type
else
  write_error('unsupported function type :                          ',token);
new(new_parameter);
new_parameter^.id := function_name;
new_parameter^.id_type := symbol_type;
new_parameter^.next := nil;
insert_symbol(function_name,symbol_type,symbol_value);
new_parameter^.address := symbol_value;
if procedure_link^.parameter_link = nil then
begin
   procedure_link^.parameter_link := new_parameter;
end
else
begin
  new_parameter^.next := procedure_link^.parameter_link;
  procedure_link^.parameter_link := new_parameter;
{   current_parameter := procedure_link^.parameter_link;
  while current_parameter^.next <> nil do current_parameter := current_parameter^.next;
  current_parameter^.next := new_parameter; }
end;
no_local_variable[procedure_level] := no_local_variable[procedure_level]+1;
```

```
  local_variable[procedure_level,no_local_variable[procedure_level]] := function_name;
{ current_parameter := procedure_link^.parameter_link;
  writeln(outfile,'beginning of listing of parameter inside procedure');
  while current_parameter <> nil do
  begin
    writeln(outfile,'current_parameter id      : ',current_parameter^.id);
    writeln(outfile,'current_parameter id_type : ',current_parameter^.id_type);
    writeln(outfile,'current_parameter address : ',current_parameter^.address);
    current_parameter := current_parameter^.next;
  end; }
  fetch_token;
  verify_token(token,semicolon);
  fetch_token;
  program_main_block;
  verify_token(token,semicolon);
  if no_local_variable[procedure_level] > max_local_variable then
    write_error('maximum number of local variable exceeded limit   ',token);
  for i := 1 to no_local_variable[procedure_level] do
  begin
{   writeln(outfile,'delete from symbol table : "',local_variable[procedure_level,i],'"'); }
    delete_symbol(local_variable[procedure_level,i]);
  end;
  if write_lookahead_buffer[1].id <> blank_token then generate_Nop;
  writeln(outfile,';',program_counter,': return');
  microcode_address := program_counter;
  AM2910_opcode := CRTN;
  branch_opcode := unconditional;
  output_microcode_field;
  program_counter := program_counter + 1;
  fetch_token;
  for i := 0 to max_index_register do index_register[i] := blank_token;
  inside_function_block[procedure_level] := false;
  procedure_level := procedure_level-1;
  writeln(outfile,'; end of function ',function_name,' }' );
end; { of function_main_block }
```

```
File: STDPROCD.DEF
public stdprocd;
  Procedure procedure_store_function;
  Procedure procedure_store_window;
  procedure procedure_read_function;
  Procedure procedure_gt_ffs;
  Procedure standard_procedure_block;
```

```
File: STDPROCD.PAS
module stdprocd;
$include(stdprocd.def)
$include(global.def)
$include(utility.def)
$include(exprsion.def)
$include(exprtree.def)
$include(code_gen.def)
$include(io.def)
$include(fetch_tk.def)
$include(arith.def)
$include(symbol_t.def)
$include(bit_func.def)
$include(emu_lib.def)
private stdprocd;

Procedure procedure_store_function;
var
    x,fx : operand_type;
begin
  fetch_token;
  verify_token(token,open_parenthesis);
  fetch_parameter(x); check_operand_type(x,real_symbol_type);
  verify_token(token,comma);
  fetch_parameter(fx); check_operand_type(fx,real_symbol_type);
  verify_token(token,close_parenthesis);
  find_symbol(fx.id,fx.id_type,fx.id_address,found);
  if not found then
    write_error('unknow id:                                  ',fx.id);
  fx.index_address := assign_index(fx.index);
  find_symbol(x.id,x.id_type,x.id_address,found);
  if not found then
    write_error('unknow id:                                  ',x.id);
  x.index_address := assign_index(x.index);
  reset_microcode_field;
  clear_pipeline_stage;
  operand_string(x, temp_token);
  write(outfile,'; store_function(',temp_token,',');
  operand_string(fx, temp_token);
  writeln (outfile, temp_token, ')');
  microcode_address := program_counter;
  AR := fx.id_address + fx.offset;
  if fx.index <> blank_token then
  begin
    AIR[0] := fx.index_address;
    IA1 := 1;
  end;
  AS := 1;
  output_microcode_field;
  program_counter := program_counter+1;
  microcode_address := program_counter;
```

```
  AR := x.id_address + x.offset;
  if x.index <> blank_token then
  begin
    AIR[0] := x.index_address;
    IA1 := 1;
  end;
  AS := AR;
  AIS[0] := AIR[0];
  IA0 := IA1;
  output_microcode_field;
  program_counter := program_counter+1;
  microcode_address := program_counter;
  write_opcode := store_function_opcode;
  output_microcode_field;
  program_counter := program_counter+1;
  stack_pointer := stack_pointer - 2;
end; { of store_function }

Procedure procedure_store_window;
var window : longint;
    t1,t2 : operand_type;
    i : longint;
begin
  fetch_token;
  verify_token(token,open_parenthesis);
  fetch_token;
  if symbol_type <> integer_constant_symbol_type then
    write_error('constant expected :                              ',token);
  window := integer_constant_value;
  find_symbol(token,symbol_type,symbol_value,found);
  if not found then
  begin
    insert_symbol(token,symbol_type,symbol_value);
    declare_constant(symbol_value,symbol_type,token);
  end;
  assign_temp_parameter(t1,real_symbol_type);
  t1.id := token;
  assign_temp_parameter(t2,integer_symbol_type);
  generate_ALU_operation(t2,t1,zero_operand,unary_round);
  clear_pipeline_stage;
  writeln(outfile,'; store_window(',window,')');
  microcode_address := program_counter;
  delete(t2.id,1,1);
  val_integer(t2.id,AR,i);
  AS := AR;
  output_microcode_field;
  program_counter := program_counter + 1;
  microcode_address := program_counter;
  write_opcode := store_window_opcode;
  output_microcode_field;
  program_counter := program_counter+1;
```

```
  fetch_token;
  verify_token(token,close_parenthesis);
end; { of procedure_store_window }

procedure procedure_read_function;
var
    x,fx : operand_type;
    function_number : longint;
begin
  fetch_token;
  verify_token(token,open_parenthesis);
  fetch_token;
  if symbol_type <> integer_constant_symbol_type then
    write_error('integer constant expected :                      ',token);
  function_number := integer_constant_value;
  if (function_number < 0) or (function_number > 1) then
    write_error(' function out of range :                         ',token);
  fetch_token;
  constant_assignment_type := real_constant_symbol_type;
  verify_token(token,comma);
  fetch_parameter(x); check_operand_type(x,real_symbol_type);
  verify_token(token,comma);
  fetch_parameter(fx); check_operand_type(fx,real_symbol_type);
  verify_token(token,close_parenthesis);
  find_symbol(fx.id,fx.id_type,fx.id_address,found);
  if not found then
    write_error('unknow id:                                       ',fx.id);
  fx.index_address := assign_index(fx.index);
  find_symbol(x.id,x.id_type,x.id_address,found);
  if not found then
    write_error('unknow id:                                       ',x.id);
  x.index_address := assign_index(x.index);
  reset_microcode_field;
  clear_pipeline_stage;
  operand_string(x, temp_token);
  write(outfile,'; read_function(',temp_token,',');
  operand_string(fx, temp_token);
  writeln (outfile, temp_token, ')');
  microcode_address := program_counter;
  AR := x.id_address + x.offset;
  if x.index <> blank_token then
  begin
    AIR[0] := x.index_address;
    IA1 := 1;
  end;
  AS := AR;
  IA0 := IA1;
  output_microcode_field;
  program_counter := program_counter + 1;
  microcode_address := program_counter;
  AF[0] := fx.id_address + fx.offset;
```

```
  if x.index <> blank_token then
  begin
    AIF[0] := fx.index_address;
    IA2[0] := 1;
  end;
  write_opcode :=  read_function_opcode + function_number;
  output_microcode_field;
  program_counter := program_counter + 1;
end; { of procedure_function_read }

procedure procedure_gt_ffs;
var f,r,s : operand_type;
    function_number : longint;
begin
  fetch_token;
  verify_token(token,open_parenthesis);
  fetch_token;
  if symbol_type <> integer_constant_symbol_type then
     write_error(token,'function number expected                    ');
  function_number := integer_constant_value;
  fetch_token;
  verify_token(token,comma);
  fetch_operand(f);
  fetch_token;
  verify_token(token,comma);
  fetch_parameter(r);
  verify_token(token,comma);
  fetch_parameter(s);
  write(outfile,';');
  operand_string(f, temp_token);
  write_token (outfile, temp_token);
  write( ' := GT_FFS(', function_number, ',');
  operand_string(r, temp_token);
  write_token (outfile, temp_token);
  write (',');
  operand_string(s, temp_token);
  write_token (outfile, temp_token);
  writeln (')');
  verify_token(token,close_parenthesis);
  find_symbol(f.id,f.id_type,f.id_address,found);
  if not found then
    write_error('unknown id:                                  ',f.id);
  f.index_address := assign_index(f.index);
  find_symbol(r.id,r.id_type,r.id_address,found);
  if not found then
    write_error('unknown id:                                  ',r.id);
  r.index_address := assign_index(r.index);
  find_symbol(s.id,s.id_type,s.id_address,found);
  if not found then
    write_error('unknown id:                                  ',s.id);
  s.index_address := assign_index(s.index);
```

```
  clear_pipeline_stage;
  reset_microcode_field;
  microcode_address := program_counter;
  AR := r.id_address + r.offset;
  if r.index <> blank_token then
  begin
    AIR[0] := r.index_address;
    IA1 := 1;
  end;
  AS := s.id_address + s.offset;
  if s.index <> blank_token then
  begin
    AIS[0] := s.index_address;
    IA0 := 1;
  end;
  output_microcode_field;
  program_counter := program_counter+1;
  microcode_address := program_counter;
  AF[0] := f.id_address + f.offset;
  if f.index <> blank_token then
  begin
    AIF[0] := f.index_address;
    IA2[0] := 1;
  end;
  msw := word_and (function_number, 1);
  write_opcode := word_shr(function_number, 1);
  output_microcode_field;
  program_counter := program_counter+1;
end; { of procedure_gt_ffs }

Procedure standard_procedure_block;
begin
  if (token = send) or (token = send_msw) or (token = send_lsw) then
  begin
    send_procedure;
  end
  else
  if (token = receive) or (token = receive_msw) or (token = receive_lsw) then
  begin
    receive_procedure;
  end
  else
  if token = store_function then
    procedure_store_function
  else
  if token = store_window then
    procedure_store_window
  else
  if token = read_function then
    procedure_read_function
  else
```

```
  if token = proc_reset then
  begin
  end
  else
  if token = gt_ffs_token then
  begin
    procedure_gt_ffs
  end;
  fetch_token;
end; { of standard_procedure_block }.
```

```
File: STRI.PAS
program stri (input, output);
const
   blank_token = '                                                  ';

type
   token_type = packed array [1..50] of char;

var
   j, p : integer;
   i : longint;
   striin, striout : text;
   l : token_type;

procedure str_integer ( l : longint; var token : token_type);
var
   i, j, k : integer;
   temp_char : char;

begin
   token := blank_token;
   i := 1;
   j := 1;
   if l = 0 then
      begin
         k := 2;
         token[1] := '0'
      end
   else
      begin
         if l < 0 then
            begin
               token[i] := '-';
               i := i + 1
            end;
         j := i;
         l := abs(l);
         writeln(' l = ',l);
         while l > 0 do
            begin
               token[i] := chr ((l mod 10) + 30H);
               writeln ( 'token[',i,'] := ', token[i]);
               l := l div 10;
               i := i + 1
            end;
         k := i;
         i := i - 1;
         while j < i do
            begin
               Temp_Char := token[i];
               token[i] := token[j];
```

```
                token[j] := temp_char;
                j := j + 1;
                i := i - 1
             end;
       end;
end; { str_integer }


 begin
   rewrite (striout, ':$:striout');
   reset (striin, ':$:striin');
   while not eof(striin) do
   begin
   readln (striin, i);
   str_integer (i,l);
   j := 0;
   for j := 1 to 50 do
      write (striout,l[j]);
   writeln(striout)
   end
end.
```

File: STRR.PAS

```
program strr (input, output);
const
   blank_token ='                                                  ';

type
   token_type = packed array [1..50] of char;

var
   j, p : integer;
   i : real;
   x : token_type;
   strrin, strrout : text;


function x_to_the_y  ( x : real ; y : integer ) : real;

var
  i : integer;
  total : real;

begin
   total := 1;
   for i := 1 to y do
      total := total * x;
   x_to_the_y := total
end; { x_to_the_y }

procedure str_integer ( l : longint; var token : token_type);
var
   i, j, k : integer;
   temp_char : char;

begin
   token := blank_token;
   i := 1;
   j := 1;
   if l = 0 then
      begin
         k := 2;
         token[1] := '0'
      end
   else
      begin
         if l < 0 then
            begin
               token[i] := '-';
               i := i + 1
            end;
         j := i;
         l := abs(l);
```

```
         writeln(' l = ',l);
         while l > 0 do
            begin
               token[i] := chr ((l mod 10) + 30H);
               writeln ( 'token[',i,'] := ', token[i]);
               l := l div 10;
               i := i + 1
            end;
         k := i;
         i := i - 1;
         while j < i do
            begin
               Temp_Char := token[i];
               token[i] := token[j];
               token[j] := temp_char;
               j := j + 1;
               i := i - 1
            end;
      end;
end; { str_integer }

procedure str_real ( x : real; var string : token_type );

var
   base, mantissa, fraction : real;
   i, j, exponent  : integer;
   temp_string : token_type;

begin
   string := blank_token;
   if x = 0 then
      exponent := 0
   else
      begin
         exponent := ltrunc (ln (abs(x))/ln(10));
         if (exponent < 1) and (abs(x) < 1) then
            exponent := exponent -1
      end;
   if exponent < 1 then
      base := 10
   else
      base := 0.1;
   mantissa := x * (x_to_the_y (base, abs(exponent)));
   str_integer (ltrunc (mantissa), temp_string);
   i := 1;
   string[i] := temp_string[i];
   if temp_string[2] <> ' ' then
      begin
         i:= i + 1;
         string[i] := temp_string[i]
      end;
```

```
    i := i + 1;
    string[i] := '.';
    fraction := abs(mantissa);
    for j := 1 to 10 do
       begin
          fraction := fraction - ltrunc(fraction);
          fraction := fraction * 10;
          str_integer (ltrunc(fraction), temp_string);
          string[i + j] := temp_string[1];
       end;
    i := i + j + 1;
    string[i] := 'e';
    str_integer (exponent, temp_string);
    for j := 1 to 3 do
      string[i + j] := temp_string[j];
end; { str_real }

 begin
   rewrite (strrout, ':$:strrout');
   reset (strrin, ':$:strrin');
   while not eof(strrin) do
   begin
   readln (strrin, i);
   str_real (i,x);
   j := 0;
   for j := 1 to 50 do
      write (strrout,x[j]);
   writeln(strrout)
   end
end.
```

```
File: SYMBOL_T.DEF
public symbol_t;
  Procedure delete_symbol(symbol_name:token_type);
  Procedure find_symbol(symbol_name:token_type;var symbol_type,
                                  symbol_value:longint; var found : boolean);
  Procedure insert_symbol(symbol_name:token_type;symbol_type:integer;
                                                  var symbol_value :longint);
```

File: SYMBOL_T.PAS

```
module symbol_t;
public symbol_t;
$include(global.def)
$include(fetch_tk.def)
$include(utility.def)
$include(bit_func.def)
$include(emu_lib.def)
$include(symbol_t.def)
private symbol_t;

function hash(token : token_type):longint;
var g,h,i : longint;
begin
  h := 0;
  for i := 1 to length(token) do
  begin
    h := word_shl(h, 4) + ord(token[i]);
    g := word_and(h, 0f000H);
{! 12. Assign unsigned values of $8000 or larger only to Word or LongInt types.}
    if g <> 0 then h := word_xor(h, word_shr (g, 12));
  end;
  if h < 0 then h := word_shr(h, 1);
  h := h mod prime;
  hash := h;
end;

Procedure delete_symbol;
var h : longint;
begin
  h := hash(symbol_name);
  found := false;
  (* search the list for a match to input name *)
  current_symbol := first_symbol[h];       (* start search at head of list *)
  if symbol_name = first_symbol[h]^.name then
  begin
    if first_symbol[h]^.next = nil then
    begin
      dispose(first_symbol[h]);
      first_symbol[h] := nil;
    end
    else
    begin
      first_symbol[h] := current_symbol^.next;
      dispose(current_symbol);
    end;
  end
  else
  begin
    While (current_symbol^.next <> nil) and (not found) do
    Begin
```

```
      found := (symbol_name = current_symbol^.next^.name);  (* is symbol stored in this record ? *)
      if not found then
        current_symbol := current_symbol^.next   (* advance if not found *)
    end; (* of while *)
    if found then
    begin
      (* delete symbol from the entry *)
      new_symbol := current_symbol^.next;
      current_symbol^.next := new_symbol^.next;
      dispose(new_symbol);
    end
    else
    begin
      write_error('symbol to be deleted not found :                  ',symbol_name);
    end;
  end;
end; { of delete_symbol }

Procedure find_symbol(symbol_name:token_type;var symbol_type,symbol_value:
                                              longint; var found : boolean);
label 1;
var h : longint;
    current_symbol :  symbol_pointer;
    old_symbol_type : longint;
    dummy_integer : longint;
begin
  if (symbol_name[1] = '#') or (symbol_name[1] = '&') then
  begin
    symbol_name[1] := '0';
    val_integer(symbol_name,symbol_value,h);
    if h <> 0 then write_error('Invalid symbol search :                          ',symbol_name);
    found := true;
    goto 1; { exit }
  end;
  h := hash(symbol_name);
  found := false;
  old_symbol_type := symbol_type;
  symbol_type := general_symbol_type;
  current_symbol := first_symbol[h];       (* start search at head of list *)
  (* search the list for a match to input name *)
  While (current_symbol <> nil) and (not found) do
  Begin
    found := (symbol_name = current_symbol^.name);  (* is symbol stored in this record ? *)
    if not found then
    begin
{      writeln(outfile,'find symbol : ',current_symbol^.name,' ---> ',current_symbol^.value); }
      current_symbol := current_symbol^.next   (* advance if not found *)
    end;
  end; (* of while *)
  if found then
  begin
```

```
    (* return value that was stored in table *)
    symbol_value := current_symbol^.value;
    symbol_type  := current_symbol^.symbol_type;
    if (symbol_type = procedure_symbol_type) or
       (symbol_type = function_symbol_type) then
      procedure_link := current_symbol;
    if symbol_type = integer_constant_symbol_type then
      integer_constant_value := lround(current_symbol^.constant_value)
    else
    if symbol_type = real_constant_symbol_type then
      real_constant_value := current_symbol^.constant_value;
  end
  else
  begin
    { restore the symbol_type if not found }
    symbol_type := old_symbol_type;
  end;
1: { exit }
end; { of find_symbol }

Procedure insert_symbol;
var h : longint;
   found : boolean;
begin
{ writeln(outfile,'----- beginning of insert_symbol ------ ');
  writeln(outfile,'      inserting    : "',symbol_name,'"'); }
  h := hash(symbol_name);
  found := false;
  new(new_symbol);
  new_symbol^.next := first_symbol[h];
  new_symbol^.parameter_link := nil;
  new_symbol^.name := symbol_name;
  new_symbol^.symbol_type := symbol_type;
  new_symbol^.scope := procedure_level;
  first_symbol[h] := new_symbol;
  current_symbol := first_symbol[h]^.next;
  while current_symbol <> nil do
  begin
{   writeln(outfile,'symbol^.name : ',current_symbol^.name); }
    if current_symbol^.name = symbol_name then
    begin
      if current_symbol^.scope = procedure_level then
      begin
        write_error('duplicate id                                      ',symbol_name);
      end;
    end;
    current_symbol := current_symbol^.next;
  end;
  if (symbol_type = real_symbol_type) then
  begin
    new_symbol^.value := next_dataram_location;
```

```
    symbol_value := new_symbol^.value;
    write(outfile,'; ');
    write_token (outfile, symbol_name);
    writeln (outfile, ' ---> ',symbol_value,' (real)');
  end
  else
  if (symbol_type = boolean_symbol_type) then
  begin
    new_symbol^.value := next_dataram_location;
    symbol_value := new_symbol^.value;
    write(outfile,'; ');
    write_token (outfile, symbol_name);
    writeln (outfile,' ---> ',symbol_value,' (boolean)')
  end
  else
  if (symbol_type = real_constant_symbol_type) then
  begin
    new_symbol^.value := next_dataram_location;
    symbol_value := new_symbol^.value;
    new_symbol^.constant_value := real_constant_value;
    write(outfile,'; ');
    write_token (outfile, symbol_name);
    writeln (outfile,' ---> ',symbol_value,' (real constant)');
  end
  else
  if (symbol_type = boolean_constant_symbol_type) then
  begin
    new_symbol^.value := next_dataram_location;
    symbol_value := new_symbol^.value;
    new_symbol^.constant_value := real_constant_value;
    write(outfile,'; ');
    write_token (outfile, symbol_name);
    writeln (outfile,' ---> ',symbol_value,' (boolean constant)');
  end
  else
  if (symbol_type = integer_symbol_type) then
  begin
    new_symbol^.value := next_dataram_location;
    symbol_value := new_symbol^.value;
    write(outfile,'; ');
    write_token (outfile, symbol_name);
    writeln (outfile,' ---> ',symbol_value,' (integer)');
  end
  else
  if (symbol_type = integer_constant_symbol_type) then
  begin
    new_symbol^.value := next_dataram_location;
    symbol_value := new_symbol^.value;
    new_symbol^.constant_value := integer_constant_value;
    i := next_dataram_location; { advance dataram address }
    write(outfile,'; ');
```

```
    write_token (outfile, symbol_name);
    writeln (outfile,' ---> ',symbol_value,' (integer constant)');
  end
  else
  if (symbol_type = real_array_symbol_type) then
  begin
    symbol_value := dataram_address - array_lower_range;
    new_symbol^.value := symbol_value;
    dataram_address := dataram_address + (array_upper_range-array_lower_range)
                       + 1;
    i := next_dataram_location; { advance dataram address }
    write(outfile,'; ');
    write_token (outfile, symbol_name);
    writeln (outfile,' ---> ',symbol_value,' (real array)');
  end
  else
  if (symbol_type = integer_array_symbol_type) then
  begin
    symbol_value := dataram_address - array_lower_range;
    new_symbol^.value := symbol_value;
    dataram_address := dataram_address + (array_upper_range-array_lower_range)
                       + 1;
    i := next_dataram_location; { advance dataram address }
    write(outfile,'; ');
    write_token (outfile, symbol_name);
    writeln (outfile,' ---> ',symbol_value,' (integer array)');
  end
  else
  if (symbol_type = boolean_array_symbol_type) then
  begin
    symbol_value := dataram_address - array_lower_range;
    new_symbol^.value := symbol_value;
    dataram_address := dataram_address + (array_upper_range-array_lower_range)
                       + 1;
    i := next_dataram_location; { advance dataram address }
    write(outfile,'; ');
    write_token (outfile, symbol_name);
    writeln (outfile,' ---> ',symbol_value,' (boolean array)');
  end
  else
  if (symbol_type = procedure_symbol_type) or
     (symbol_type = function_symbol_type) then
  begin
    new_symbol^.parameter_link := nil;
    new_symbol^.value := symbol_value;
    procedure_link := new_symbol;
    write(outfile,'; ');
    write_token (outfile, symbol_name);
    writeln (outfile,' ---> ',symbol_value,' (procedure/function)');
  end
  else
```

```
  if (symbol_type = label_symbol_type) then
  begin
    new_symbol^.value := symbol_value;
    write(outfile,'; ');
    write_token (outfile, symbol_name);
    writeln (outfile,' ---> ',symbol_value,' (label)');
  end;
{ writeln(outfile,'insert symbol : ',new_symbol^.name,' ----> ',i
                                                     new_symbol^.value);}
end; (* of insert_symbol *).
```

```
File: UTILITY.DEF
public utility;
  procedure assign_stack_operand(var operand : operand_Type;op_type:longint);
  Procedure verify_token(token,expected_token : token_type);
  procedure error_found;
  procedure write_error(error_message,token : token_type);
  Function next_dataram_location  : longint;
  Procedure assign_percent_variable(var token : token_type);
  Function next_temp_variable_location  : longint;
  Procedure reset_temp_variable_address;
  Procedure declare_constant(ram_address,constant_type: longint;constant_id : token_type);
  procedure operand_string(operand : operand_type; var token : token_type);
  procedure check_operand_type(operand : operand_type;symbol_type:longint);
  procedure reset_stack_pointer;
  procedure free_stack_operand;
  procedure decrement_stack_pointer;
  procedure assign_temp_variable(var variable : token_type);
  procedure reset_operand(var operand : operand_type);
  procedure simplify_type(var symbol_type:longint);
  procedure assign_dummy_parameter(var parameter:operand_type);
  procedure assign_parameter(var parameter: operand_type;
                                                    parameter_type :longint);
  procedure assign_temp_parameter(var parameter: operand_type;
                                                   parameter_type : longint);
  procedure assign_parametric_operand(var parameter: operand_type);
  procedure fetch_expression(var F,R,S : operand_type; expression : expression_pointer);
  procedure store_expression(var F,R,S : operand_type; expression : expression_pointer);
  procedure clear_index_register;
  procedure clear_temp_index;
```

```
File: UTILITY.PAS
module utility;
$include(pqclose.def)
$include(utility.def)
$include(global.def)
$include(emu_lib.def)
$include(ieee_cnv.def)
public UDI;
   procedure dqexit (completion_code : word);
private utility;

procedure clear_temp_index;
var i : integer;
begin
  for i := 0 to max_index_register do
  begin
    if index_register[i][1] = '&' then
      index_register[i] := blank_token;
  end;
end;

procedure clear_index_register;
var i : integer;
begin
  for i := 0 to max_index_register do
  begin
    index_register[i] := blank_token;
  end;
end;

procedure assign_stack_operand(var operand : operand_Type;op_type:longint);
begin
  str_integer(stack_pointer,operand.id);
  temp_token := blank_token;
  temp_token[1] := '#';
  concat (temp_token, operand.id);
  operand.id := temp_token;
  operand.id_type := op_type;
  operand.index := blank_token;
  operand.offset := 0;
  decrement_stack_pointer;
end; { assign_stack_operand }

Procedure verify_token(token,expected_token : token_type);
begin
  if token <> expected_token then
  begin
    writeln(errorfile);
    writeln(errorfile,'!!!! syntax error ,"',token,'" received');
    writeln(errorfile,'                    "',expected_token,'" expected');
    error_found;
```

```
  end;
end; { of verify_token }

procedure error_found;
var
  completion_code :word;
begin
  gotoxy(5,12);
  write('Check error at : ');
  write_token (output, error_filename);
  gotoxy(5,24);
  pqclose (errorfile);
  pqclose (outfile);
  pqclose (constant_file);
  dqexit(completion_code) { halt }
end; { of error_found }

procedure write_error ( error_message, token : token_type);
var
  completion_code :word;
begin
  writeln(errorfile);
  write(errorfile,'!!! Error, ');
  write_token (errorfile, error_message);
  write (errorfile,'"');
  write_token (errorfile, token);
  writeln (errorfile, '"');
  gotoxy(5,12);
  write('Check error at : ');
  write_token (output, error_filename);
  gotoxy(5,24);
  pqclose (errorfile);
  pqclose (outfile);
  pqclose (constant_file);
  dqexit(completion_code) { halt }
end;

Function next_dataram_location  : longint;
Begin
  if dataram_address < dataram_address_limit - temp_variable_limit then
  begin
    next_dataram_location := dataram_address;    (* return memory spot *)
    dataram_address := dataram_address+1;    (* point to next spot *)
{   if (symbol_type = integer_symbol_type) or
       (symbol_type = integer_constant_symbol_type) then
       dataram_address := dataram_address+1;  }
  end
  else
  begin
    writeln(errorfile);
    writeln(errorfile,'Data ram memory Overrun, too many variables');
```

```
    writeln(errorfile,'Maximum number of variables allowed '
                                                    ,dataram_address_limit);
    error_found;
  end;
end; { of next_dataram_location }

procedure assign_percent_variable(var token : token_type);
var i : longint;
temp_string : token_type;
begin
  temp_string := blank_token;
  token := blank_token;
  percent_variable_counter := percent_variable_counter + 1;
  str_integer (percent_variable_counter,token);
  temp_string[1] := '%';
  concat (temp_string, token);
  token := temp_string;
end; { of assign_percent_variable }

Function next_temp_variable_location  : longint;
Begin
  if inside_function_block[procedure_level] = false then
  begin
    if temp_variable_address < stack_pointer then
    begin
      next_temp_variable_location := temp_variable_address;    (* return memory spot *)
      temp_variable_address := temp_variable_address+1;    (* point to next spot *)
    end
    else
    begin
      writeln(errorfile);
      writeln(errorfile,'Data ram memory Overrun, too many variables');
      writeln(errorfile,'Maximum number of variables allowed '
                                                    ,dataram_address_limit);
      error_found;
    end;
  end
  else
  begin
    next_temp_variable_location := next_dataram_location;
  end;
end; { of next_temp_variable_location }

Procedure reset_temp_variable_address;
begin
{ if write_lookahead_buffer[0].id[1] = '#' then
     write_lookahead_buffer[0].id := '';  }
  temp_variable_address := dataram_address_limit - temp_variable_limit +1;
  {note that the location dataram_address_limit-temp_variable_limit is reserved
   for temporary boolean variable to avoid conflict with the pipeline assignment}
end; { of reset_temp_variable_address }
```

```
Procedure declare_constant(ram_address,constant_type:
                                             longint;constant_id : token_type);
var msw,lsw : word;
begin
  if (constant_type = real_constant_symbol_type) or
     (constant_type = boolean_constant_symbol_type) then
  begin
    Real_to_IEEE(real_constant_value,msw,lsw);
    writeln (constant_file,';declare constant ', ram_address,
                                  ' ---> ',real_constant_value,' (real)');
    writeln(constant_file,'v ',ram_address,' ',msw,' ',lsw);
  end
  else
  if constant_type = integer_constant_symbol_type then
  begin
    real_constant_value := integer_constant_value;
    Real_to_IEEE(real_constant_value,msw,lsw);
    writeln(constant_file,';declare constant ',ram_address,' ---> '
                                    ,real_constant_value:10:9,' (real/integer)');
    writeln(constant_file,'v ',ram_address,' ',msw,' ',lsw);
  end
  else
  begin
    write_error('unknown constant type                              ',token);
  end;
end;

procedure operand_string(operand : operand_type; var token : token_type);
var
    offset_string : token_type;
begin
  token := blank_token;
  token := operand.id;
  if (operand.index <> blank_token) then
  begin
    if operand.offset <> 0 then
    begin
      str_integer(operand.offset,offset_string);
      add_char_to_string (token, '[');
      concat (token, operand.index);
      add_char_to_string (token, '+');
      concat (token, offset_string);
      add_char_to_string (token, ']')
    end
    else
      begin
        add_char_to_string (token, '[');
        concat (token, operand.index);
        add_char_to_string (token, ']')
      end
```

```
  end
  else if operand.offset <> 0 then
  begin
    str_integer(operand.offset,offset_string);
    add_char_to_string (token, '[');
    concat (token, offset_string);
    add_char_to_string (token, ']')
  end
end; { of operand_string }

procedure check_operand_type(operand : operand_type;symbol_type:longint);
begin
  if operand.id_type = real_constant_symbol_type then operand.id_type :=
                                                        real_symbol_type;
  if operand.id_type = integer_constant_symbol_type then operand.id_type :=
                                                     integer_symbol_type;
  if operand.id_type <> symbol_type then
    write_error('type mismatch :                                ',operand.id);
end; { of check_operand_type }

procedure reset_stack_pointer;
begin
  stack_pointer := dataram_address_limit;
end; { of reset_stack_pointer }

procedure free_stack_operand;
begin
  stack_pointer := stack_pointer + 1;
end; { of increment_stack_pointer }

procedure decrement_stack_pointer;
begin
  stack_pointer := stack_pointer - 1;
end; { of decrement_stack_pointer }

procedure assign_temp_variable(var variable : token_type);
var
   temp_string : token_type;

begin
  temp_string := blank_token;
  str_integer (next_temp_variable_location,variable);
  temp_string[1] := '#';
  concat(temp_string ,variable);
  variable := temp_string;
end;

procedure reset_operand(var operand : operand_type);
begin
  operand.id := blank_token;
  operand.index := blank_token;
```

```
  operand.offset := 0;
  operand.id_address := 0;
end; { of reset operand }

procedure simplify_type(var symbol_type:longint);
begin
  case symbol_type of
    integer_constant_symbol_type: symbol_type := integer_symbol_type;
    real_constant_symbol_type:    symbol_type := real_symbol_type;
    boolean_constant_symbol_type: symbol_type := boolean_symbol_type;
    real_array_symbol_type:       symbol_type := real_symbol_type;
    integer_array_symbol_type:    symbol_type := integer_symbol_type;
    boolean_array_symbol_type:    symbol_type := boolean_symbol_type;
  end;
end; { simplified_data_type }

procedure assign_dummy_parameter(var parameter:operand_type);
begin
  reset_operand(parameter);
  parameter.id := blank_token;
  parameter.id[1] := '#';
end; { of assign_dummy_parameter }

procedure assign_parameter(var parameter: operand_type;parameter_type:longint);
var integer_address : longint; { used to increment the dataram address for integer }
begin
  reset_operand(parameter);
  parameter.id_type := parameter_type;
  str_integer (next_dataram_location,parameter.id);
  if parameter.id_type = integer_symbol_type then
    integer_address := next_dataram_location;
  temp_token := blank_token;
  temp_token[1] := '&';
  concat (temp_token, parameter.id);
  parameter.id := temp_token
end; { of assign_parameter }

procedure assign_temp_parameter(var parameter: operand_type;
                                                     parameter_type : longint);
begin
  reset_operand(parameter);
  parameter.id_type := parameter_type;
  str_integer (next_temp_variable_location,parameter.id);
  temp_token := blank_token;
  temp_token[1] := '#';
  concat (temp_token, parameter.id);
  parameter.id := temp_token
end; { of assign_temp_parameter }

procedure assign_parametric_operand(var parameter: operand_type);
begin
```

```
  reset_operand(parameter);
  parameter.id_address := next_dataram_location;
  str_integer (parameter.id_address,parameter.id);
  temp_token := blank_token;
  temp_token[1] := '#';
  concat (temp_token, parameter.id);
  parameter.id := temp_token
end; { of assign_parametric_operand }

procedure fetch_expression(var F,R,S : operand_type;
                                             expression : expression_pointer);
begin
  if expression^.left^.id = blank_token then
    R := zero_operand
  else
  begin
    R.id := expression^.left^.id;
    R.index := expression^.left^.index;
    R.offset := expression^.left^.offset;
    R.id_type := expression^.left^.id_type;
  end;
  if expression^.id = blank_token then
    S := zero_operand
  else
  begin
    S.id := expression^.id;
    S.index := expression^.index;
    S.offset := expression^.offset;
    S.id_type := expression^.id_type;
  end;
  F.id := expression^.left^.up^.id;
  F.index := expression^.left^.up^.index;
  F.offset := expression^.left^.up^.offset;
  F.id_type := expression^.left^.up^.id_type;
  if (F.id[1] = '%') then
  begin
    assign_temp_parameter(F,F.id_type);
    expression^.left^.up^.id := F.id;
  end;
end; { of fetch_expression }

procedure store_expression(var F,R,S : operand_type; expression : expression_pointer);
begin
  expression^.left^.id := R.id;
  expression^.left^.index := R.index;
  expression^.left^.offset := R.offset;
  expression^.left^.id_type := R.id_type;
  expression^.id := S.id;
  expression^.index := S.index;
  expression^.offset := S.offset;
  expression^.id_type := S.id_type;
```

```
  expression^.left^.up^.id := F.id;
  expression^.left^.up^.index := F.index;
  expression^.left^.up^.offset := F.offset;
  expression^.left^.up^.id_type := F.id_type;
end; { of store_expression }.
```

File: VALR.PAS

```
program valr (input, output);

const max_token_length = 50;
type tokentype = array [1..50] of char;
var
   x,i : integer;
   r : real;
   string : tokentype;
   valrout, valrin : text;

procedure val_real (charstring : tokentype; var real_number : real;
                                              var error_integer : integer);
label 1;

var i,j       : integer;
    exponent  : integer;
    x         : real;
    sign      : integer;
    exponent_token,token   : tokentype;

{   Discarding leading zero   }

Procedure Delete_leading_zero (var token : tokentype);
var i,j : integer;
begin
    i := 1;
    while i <= max_token_length do
    begin
      if (token[i] = '0') then
      begin
         for j := i to max_token_length-1 do
         begin
             token[j] := token[j+1];
         end;
         token[max_token_length] := ' ';
      end
      else
         if (token[i] = '+') or (token[i] = '-') then
             i := i + 1
         else
             i := max_token_length + 1;
   end;
end;


{ This function converts a packed array of character that represents an integer
  to real number }

function char_integer_to_real(token : tokentype): real;
label 1:
```

```
var i : integer;
    x,power : real;
begin
    x := 0;

    {check for valid integer}
    for i := 1 to max_token_length do
    begin
        if (token[i] <> ' ') and (token[i] <> '+') and (token[i] <> '-') then
            if ((ord(token[i]) < 48) or (ord(token[i]) > 57)) then
            begin
               error_integer := 9999;       { for error checking - T.F.}
               goto 1
            end;
    end;

    i := max_token_length;
    while (i > 0) and (token[i] = ' ') do i := i-1;
    if (i = 0) then goto 1;            { if token = blank token }

    x := ord(token[i])-48;
    power := 1;
    i := i-1;

    while (i > 0) do
    begin
      if ((ord(token[i]) >= 48) and (ord(token[i]) <= 57)) then
      begin
         power := power*10.0;
         x := x + (ord(token[i])-48)*power;
      end;
      if token[i] = '-' then x := -x;
      i := i-1;
    end;

1:  char_integer_to_real := x;
end;   { of char integer to real conversion }


{ beginning of the function token to real converter }

begin
    token := charstring;
    exponent := 0;
    error_integer := 0;  { if stays 0 then no error occurred - T.F. }
    delete_leading_zero (token);

{ Detecting whether digit left of decimal point is 0 }

   if (token[1] = '.' ) or ((token[2] = '.') and
                          not (token[1] in ['1'..'9'])) then
```

```
  { + or - sign could have preceded the decimal point }

  begin
      if token[1] = '.' then i := 2
      else i := 3;

      while token[i] = '0' do
      begin
        exponent := exponent-1;
        i := i + 1;
      end
  end;

{  Detecting decimal point }

  i := 1;
  while (i <= max_token_length) do
  begin
      if token[i] = '.' then
      begin
         for j := i to max_token_length-1 do
             token[j] := token[j+1];
         token[max_token_length] := ' ';
         i := max_token_length;
      end
      else
      if (token[i] = 'e') or (token[i] = 'E') then
         i := max_token_length
      else
      if  (ord(token[i]) >= 48) and (ord(token[i]) <= 57) then
                  exponent := exponent+1;

      i := i + 1;
  end;

{  check for exponential notation 'e' or 'E' }

  i := 1;
  while (i <= max_token_length) do
  begin
      if (token[i] = 'e') or (token[i] = 'E') then
      begin
         token[i] := ' ';
         for j := 1 to max_token_length do
         begin
             exponent_token[j] :=  ' ';
         end;

         for j := i+1 to max_token_length do
         begin
```

```
            exponent_token[j] := token[j];
            token[j] := ' ';
         end;
         x := char_integer_to_real(exponent_token);
         writeln(x);
         exponent := exponent + round(x);

         i := max_token_length + 1;
      end
      else
         i := i + 1;
   end;

   x := char_integer_to_real(token);
   while abs(x) >= 1 do x := x/10;

   if exponent > 0 then
      for i := 1 to exponent do x := x*10.0;
   if exponent < 0 then
      for i := -1 downto exponent do x := x/10.0;

1: real_number := x;

end;


begin
   rewrite (valrout, ':$:valrout');
   reset (valrin, ':$:valrin');
   while not eof(valrin) do
      begin
         for x := 1 to 50 do
            begin
               string[x] := ' ';
               if not eof(valrin) and not eoln(valrin) then
                  read (valrin, string[x]);
            end;
         readln (valrin);
         val_real (string, r, i);
         if i = 0 then writeln (valrout, r:20:10)
         else writeln (valrout, 'i:', i)
      end;
end.
```

C. Floating-Point Loader source code

```
/*
 Copyright 1990
 Georgia Tech Research Corporation
 Centennial Research Building
 Atlanta, GA 30332
*/


#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <host.h>


#define DATA_PORT 0x0c000
#define STATUS_PORT 0x0e000


unsigned short number_error;
char *value;
unsigned long base;
unsigned long limit;
unsigned long type;


/* microcode field(s) */
unsigned short program_counter;
unsigned short am2910_opcode;
unsigned short branch_address;
unsigned short branch_opcode;
unsigned short write_opcode;
unsigned short dsel;
unsigned short read_opcode;
unsigned short enf_bar;
unsigned short i4;
unsigned short i3;
unsigned short mc325;
unsigned short af;
unsigned short ar;
unsigned short as;
unsigned short msw;
unsigned short ia2;
unsigned short ia1;
unsigned short ia0;
unsigned short aif;
```

```
unsigned short air;
unsigned short ais;


void stop_processor( void )
{
        unsigned short temporary;

        temporary = 0;
        poke( base + STATUS_PORT, &temporary, sizeof( temporary ) );
        peek( base + STATUS_PORT, &temporary, sizeof( temporary ) );
        if ( ( temporary & 4 ) != 4 )
        {
                number_error++;
                printf( "ERROR: unable to stop the processor\n" );
        }
} /* stop_processor */


void start_processor( void )
{
        unsigned short temporary;

        temporary = 1;
        poke( base + STATUS_PORT, &temporary, sizeof( temporary ) );
        peek( base + STATUS_PORT, &temporary, sizeof( temporary ) );
        if ( ( temporary & 4 ) == 4 )
        {
                number_error++;
                printf( "ERROR: unable to start the processor\n" );
        }
} /* start_processor */


void reset_processor( void )
{
        stop_processor( );
        start_processor( );
} /* reset_processor */


int rfi( void )
{
        register unsigned short count;
        unsigned short temporary;

        for ( count = 0; count != 1000; count++ )
        {
                peek( base + STATUS_PORT, &temporary, sizeof( temporary ) );
                if ( ( temporary & 2 ) == 2 )
                        return( TRUE );
```

```
        }

    number_error++;
    printf("ERROR: processor RFI not responding after 1000 counts\n");

        return( FALSE );
} /* rfi */


void send( unsigned short *buffer )
{
        if ( rfi( ) )
                poke( base + DATA_PORT, buffer, sizeof( *buffer ) );
} /* send */


int dav( void )
{
        register unsigned short count;
        unsigned short temporary;

        for ( count = 0; count != 1000; count++ )
        {
                peek( base + STATUS_PORT, &temporary, sizeof( temporary ) );
                if ( ( temporary & 1 ) == 1 )
                        return( TRUE );
        }

    number_error++;
    printf( "ERROR: processor DAV not responding after 1000 counts\n" );

        return( FALSE );
} /* dav */


void receive( unsigned short *buffer )
{
        if ( dav( ) )
                peek( base + DATA_PORT, buffer, sizeof( *buffer ) );
} /* receive */


void reset_microcode_field( void )
{
        am2910_opcode = 0x0e;
        branch_address = 0;
        branch_opcode = 0;
        write_opcode = 0;
        dsel = 0;
        read_opcode = 0;
        enf_bar = 0;
```

```
        i4 = 0;
        i3 = 0;
        mc325 = 0;
        af = 0;
        ar = 1;
        as = 1;
        msw = 0;
        ia2 = 0;
        ia1 = 0;
        ia0 = 0;
        aif = 0;
        air = 0;
        ais = 0;
} /* reset_microcode */


/* pack the instruction fields and down load them to the processor */
void load_code( void )
{
        unsigned short microcode[6];
        unsigned short index;
        unsigned short offset;
        unsigned short temporary;

        microcode[0] = (msw<<15)
                + (ia2<<14)
                + (ia1<<13)
                + (ia0<<12)
                + (aif<<8)
                + (air<<4)
                + ais;
        microcode[1] = as;
        microcode[2] = ar;
        microcode[3] = af;
        microcode[4] = (branch_opcode<<12)
                + (write_opcode<<9)
                + dsel
                + (read_opcode<<6)
                + (enf_bar<<5)
                + (i4<<4)
                + (i3<<3)
                + mc325;
        microcode[5] = (am2910_opcode<<12)
                + branch_address;

        for ( index = 0; index <=5; index++ )
        {
                offset = (index<<13) + (program_counter<<1);
                poke( base + offset, &microcode[index], sizeof( microcode[index] ) );
                peek( base + offset, &temporary, sizeof( temporary ) );
                if ( microcode[index] != temporary )
```

```
                {
                        number_error++;
                        printf( "ERROR: instruction memory loading failed:\n" );
                        printf( " write %04x \n", microcode[index] );
                        printf( " read  %04x \n", temporary );
                }
        }
} /* load_code */


/* generate codes for the processor to receive data from the host */
void generate_receive( unsigned short address )
{
        reset_microcode_field( );
        af = address;
        if ( address == 1 )
        {
                as = 0;
                ar = 0;
        }
        msw = 1;
        do
        {
                write_opcode = 0;
                am2910_opcode = 0x0e;
                load_code( );
                program_counter = program_counter + 1;
                am2910_opcode = 3;
                branch_address = program_counter;
                branch_opcode = 2;
                write_opcode = 2;
                load_code( );
                program_counter = program_counter + 1;
                msw = msw - 1;
        }
        while ( msw != -1 );
        reset_microcode_field( );
} /* generate_receive */


/* generate codes for the processor to send data from the host */
void generate_send( unsigned short address )
{
        reset_microcode_field( );
        as = address;
        ar = address;
        if ( address == 0 )
        {
                af = 1;
        }
        msw = 1;
```

```
        do
        {
                read_opcode = 0;
                am2910_opcode = 0x0e;
                load_code( );
                program_counter = program_counter + 1;
                am2910_opcode = 3;
                branch_address = program_counter;
                branch_opcode = 3;
                read_opcode = 2;
                load_code( );
                program_counter = program_counter + 1;
                msw = msw - 1;
        }
        while ( msw != -1 );
        reset_microcode_field( );
} /* generate_send */


/* load the procedure that will fetch constants from the host */
void load_pct( char *path )
{
        FILE *file;
        char line[256];
        unsigned short address, msw, lsw;

        if ( ( file = fopen( path, "r" ) ) == NULL )
        {
                printf( "ERROR: unable to open for read %s\n", path );
                exit( -1 );
        }

        stop_processor( );

        program_counter = 0;
        reset_microcode_field( );
        load_code( );
        program_counter = program_counter + 1;
        load_code( );
        program_counter = program_counter + 1;

        while ( fgets( line, sizeof( line ), file ) != NULL )
        {
                if ( line[ 0 ] == ';' )
                        continue;

                if ( sscanf( line, "v %d %d %d\n", &address, &msw, &lsw ) == 3 )
                {
                        generate_receive( address );
                        generate_send( address );
                }
```

```
	}

	am2910_opcode = 3;
	branch_opcode = 0x0c;
	branch_address = program_counter;
	load_code( );

	fclose( file );
} /* load_pct */


/* this procedure is used to load the constants to the amd floating point processor */
void load_hct( char *path )
{
	FILE *file;
	char line[256];
	unsigned short address, msw, lsw;
	unsigned short temporary;

	if ( ( file = fopen( path, "r" ) ) == NULL )
	{
		printf( "ERROR: unable to open for read '%s'\n", path );
		exit( -1 );
	}

	start_processor( );

	while ( fgets( line, sizeof( line ), file ) != NULL )
	{
		if ( line[ 0 ] == ';' )
			continue;

		if ( sscanf( line, "v %d %d %d\n", &address, &msw, &lsw ) == 3 )
		{
			send( &msw );
			send( &lsw );

			receive( &temporary );
			if ( temporary != msw )
			{
				number_error++;
				printf( "ERROR: constant load failed: msw\n" );
			}

			receive( &temporary );
			if ( temporary != lsw )
			{
				number_error++;
				printf( "ERROR: constant load failed: lsw\n" );
			}
		}
```

```
        }

        fclose( file );
} /* load_hct */


/* this program is used to load program to the amd floating point processor */
void load_fpp( char *path )
{
        FILE *file;
        char line[256];
        unsigned short offset;
        unsigned short temporary;

        stop_processor( );

        if ( ( file = fopen( path, "r" ) ) == NULL )
        {
                printf( "ERROR: unable to open for read %s\n", path );
                exit( -1 );
        }

        while ( fgets( line, sizeof( line ), file ) != NULL )
        {
                if ( line[0 ] == ';' )
                        continue;

                if ( sscanf( line, "b %d %d\n", &program_counter, &branch_address ) == 2 )
                {
                        temporary = 5;
                        offset = (temporary<<13) + (program_counter<<1);
                        peek( base + offset, &temporary, sizeof( temporary ) );
                        am2910_opcode = (temporary>>12);
                        temporary = (am2910_opcode<<12) + branch_address;
                poke( base + offset, &temporary, sizeof( temporary ) );
                }

                if ( sscanf( line, "c %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d \n",
                        &program_counter,&am2910_opcode,&branch_address,&branch_opcode,
                        &write_opcode,&dsel,&read_opcode,&enf_bar,&i4,&i3,&mc325,
                        &af,&ar,&as,&msw,&ia2,&ia1,&ia0,&aif,&air,&ais) == 21 )
                {
                        load_code( );
                }
        }

        fclose( file );
} /* load_fpp */


#define PROGRAM argument[ 0 ]
```

```
#define NAME argument[ 1 ]
#define PATH argument[ 2 ]

void main( int number_argument, char *argument[ ] )
{
	char path[ 256 ];

	initialize_environment( ":HOME:ENVIRONMENT" );

	if ( number_argument != 3 )
	{
		fprintf( stderr, "usage: %s <name> <path>\n", PROGRAM );
		exit( 0 );
	}

	if ( ( value = getenv( NAME ) ) == NULL )
	{
		fprintf( stdout, "ERROR: '%s' not found in environment\n", NAME );
		exit( -1 );
	}

	if ( sscanf( value, "%lx:%lx:%lx:", &base, &limit, &type ) != 3 )
	{
		fprintf( stdout, "ERROR: unable to parse '%s = %s'\n", NAME, value );
		exit( -1 );
	}

	number_error = 0;
	reset_processor( );

	printf( "loading %s\n", NAME );
	strcpy( path, PATH );
	strcat( path, ".hct" );
	load_pct( path );

	printf( "loading %s data\n", PATH );
	strcpy( path, PATH );
	strcat( path, ".hct" );
	load_hct( path );

	printf( "loading %s code\n", PATH );
	strcpy( path, PATH );
	strcat( path, ".fpp" );
	load_fpp( path );

	printf( "starting %s\n", NAME );
	start_processor( );

	if ( number_error != 0 )
		printf( "number error(s) = %d\n", number_error );
```

```
        exit( 0 );
} /* main */
```

D. Crossbar/Sequencer Compiler source code


```
{------------------------------------------------------------------}
{                                                                  }
{   iRMX PFP Crossbar/Sequencer Compiler V1.0                      }
{     Compatible with the GT-XB/2 and RMXII only.                  }
{                                                                  }
{                                                                  }
{   Key Words : LOOP, CYCLE                                        }
{        p(receiver)[,p(receiver)] := p(sender).(repeat factor)     }
{        comments contained between []                              }
{                                                                  }
{   BootLoadable OMF286 Absolute Code is generated for the         }
{   sequencer (sequencer.bl) and crossbar (crossbar.bl) memories.  }
{                                                                  }
{   OMF86 and RMXI by : T. S. Floyd                                }
{   Date :       April 22, 1988 Version 3.0                        }
{   OMF286 and RMXII by : T. R. Collins & S. R. Wachtel            }
{   Date :       February 22, 1990 Version 1.0                              }
{                                                                  }
{------------------------------------------------------------------}
MODULE xbc;


PUBLIC UDI;
  type
    date_time_type = record
        system_time : longint;
        date : array[0..7] of char;
        time : array[0..7] of char;
    end;
  procedure dqdecodetime(var datetime : date_time_type; var except : integer);


PUBLIC RANDOMIO;
          procedure setrandom(var f: bytes);
          procedure seekread(var f: bytes; r: longint);
          procedure seekwrite(var f: bytes; r: longint);
          function position(var f: bytes): longint;
```

```
PUBLIC xbc;

PROGRAM xbc(input,output);

const
    start_lower =       61H;
    convert_upper =     20H;
    convert_numb =      30H;
    cr =                0DH;
    lf =                0AH;
    space =             20H;
    max_comm =           16;
    max_proc_numb =      31;
    max_digits =          2;
    max_errors =         30;
    bits_in_word =       16;
    max_x_bus =          15;
    min_y_bus =          16;
    fourth_xbar =         7;
    max_step =         1024;
    name_size =          20;
    end_mark =           32;
    null =              00h;

type

    byte =          0..255;
    bus_list =      array [0..max_proc_numb] of char;
    error_array =   array [1..max_errors] of integer;
    mat_array =     array [0..max_x_bus,min_y_bus..max_proc_numb] of integer;
    proc_array =    array [1..max_comm,0..max_proc_numb] of integer;
    repeat_array =  array [1..max_comm] of integer;
    word_array =    array [0..max_proc_numb] of integer;
    digit_store =   array [1..max_digits] of char;
    proc_info = record
        repeat_factor : repeat_array;
        receiver :      proc_array;
        sender :        proc_array;
    end;
    transfer = record
        count :    integer;
        sender :   repeat_array;
        receiver : repeat_array;
    end;

var
    file_name :  packed array[1..name_size] of char;
    seq_abs,
    xbar_abs : file of byte;
    input_file,
    addr_setup,
```

```
    setup :        text;
    eof_flag,
    stop_proc,
    jump_flag,
    gen_addr,
    gen_setup  :  boolean;
    comm_count,
    cycle_number,
    step_number,
    jump_number,
    jump_cycle :  integer;
    check_sum :   byte;
    clear_proc,
    processor :   proc_info;
    error_log :   error_array;
    clear_tran,
    xtox,
    xtoy,
    ytox,
    ytoy :        transfer;
    xbar_bus :    bus_list;
    clear_matrix,
    xbar_matrix : mat_array;
    num_abs_seq_sets_output : integer;
    num_abs_xbar_sets_output : integer;


PROCEDURE open_files;

var
    input_char : char;
    kount      : integer;

begin

    writeln('iRMXII PFP Crossbar/Sequencer Compiler V1.0');
          writeln;

    for kount := 1 to name_size do
        file_name[kount] := ' ';

    write('Enter name of file with compiler input - ');
    kount := 0;
    repeat
        read(input_char);
        kount := kount + 1;
        file_name[kount] := input_char;
    until ( ord(input_char) = end_mark );
    file_name[kount] := chr(null);

    writeln;
```

```
    write('Do you want to generate the setup file, setup.dat (y/n)?');
    readln(input_char);
    if input_char in ['Y','y'] then gen_setup := true
    else gen_setup := false;

    writeln;
    write('Do you want to generate the address file, address.dat (y/n)?');
    readln(input_char);
    if input_char in ['Y','y'] then gen_addr := true
    else gen_addr := false;

    reset(input_file,file_name);
          setrandom(seq_abs);
    rewrite(seq_abs,'SEQUENCER.BL');
          setrandom(xbar_abs);
    rewrite(xbar_abs,'CROSSBAR.BL');
    if gen_setup then
        rewrite(setup,'SETUP.DAT');
    if gen_addr then begin
        rewrite(addr_setup,'ADDRESS.DAT');
        writeln(addr_setup,'Next Addresses');
        writeln(addr_setup);
        writeln(addr_setup,
                '    Cycle    Crossbar Address    Sequencer Address');
        writeln(addr_setup);
    end;

end;  { PROCEDURE open_files }



FUNCTION power( expon, base : integer ) : integer;

begin

    if expon >=1 then
        power := power( expon-1, base ) * base
    else
        power := 1;

end;  { FUNCTION power }



PROCEDURE write_byte( which_one: char; input_value: byte );

begin

    if ( which_one = 'S' )
          then
                    begin
```

```
                        seq_abs^ := input_value;
                        put( seq_abs );
                end
    else
                begin
                        xbar_abs^ := input_value;
                        put( xbar_abs );
                end;

end;  { PROCEDURE write_byte }



PROCEDURE initialize_output_files;

var
    i : integer;
    date_time : date_time_type;
    except : integer;

begin
    date_time.system_time := 0;    { Request new time and decode }
    dqdecodetime(date_time, except);

    num_abs_seq_sets_output := 0;

    write_byte( 'S', 0A2H ); { A2H = boot loadable module }

    write_byte( 'S', 00H ); {total space}
    write_byte( 'S', 00H );
    write_byte( 'S', 00H );
    write_byte( 'S', 00H );

    for i := 0 to 7 do
      write_byte( 'S', ord(date_time.date[i]) );

    for i := 0 to 7 do
      write_byte( 'S', ord(date_time.time[i]) );

    write_byte( 'S', ord( 'i' ) );       {Creator}
    write_byte( 'S', ord( 'R' ) );
    write_byte( 'S', ord( 'M' ) );
    write_byte( 'S', ord( 'X' ) );
    write_byte( 'S', ord( ' ' ) );
    write_byte( 'S', ord( 'P' ) );
    write_byte( 'S', ord( 'F' ) );
    write_byte( 'S', ord( 'P' ) );
    write_byte( 'S', ord( ' ' ) );
    write_byte( 'S', ord( 'C' ) );
    write_byte( 'S', ord( 'r' ) );
    write_byte( 'S', ord( 'o' ) );
```

```
write_byte( 'S', ord( 's' ) );
write_byte( 'S', ord( 's' ) );
write_byte( 'S', ord( 'b' ) );
write_byte( 'S', ord( 'a' ) );
write_byte( 'S', ord( 'r' ) );
write_byte( 'S', ord( '/' ) );
write_byte( 'S', ord( 'S' ) );
write_byte( 'S', ord( 'e' ) );
write_byte( 'S', ord( 'q' ) );
write_byte( 'S', ord( 'u' ) );
write_byte( 'S', ord( 'e' ) );
write_byte( 'S', ord( 'n' ) );
write_byte( 'S', ord( 'c' ) );
write_byte( 'S', ord( 'e' ) );
write_byte( 'S', ord( 'r' ) );
write_byte( 'S', ord( ' ' ) );
write_byte( 'S', ord( 'C' ) );
write_byte( 'S', ord( 'o' ) );
write_byte( 'S', ord( 'm' ) );
write_byte( 'S', ord( 'p' ) );
write_byte( 'S', ord( 'i' ) );
write_byte( 'S', ord( 'l' ) );
write_byte( 'S', ord( 'e' ) );
write_byte( 'S', ord( 'r' ) );
write_byte( 'S', ord( ' ' ) );
write_byte( 'S', ord( 'V' ) );
write_byte( 'S', ord( '1' ) );
write_byte( 'S', ord( '.' ) );
write_byte( 'S', ord( '0' ) );

write_byte( 'S', 00H );  {GDT}
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );

write_byte( 'S', 00H );  {IDT}
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );

write_byte( 'S', 00H );  {TSS}
write_byte( 'S', 00H );

write_byte( 'S', 96 );     {ABSTXT Location}
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );
```

```
write_byte( 'S', 00H );    {DEBTXT Location}
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );

write_byte( 'S', 00H );    {ENDTXT Location - MUST BE DONE LAST}
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );

write_byte( 'S', 00H );    {Next Partition}
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );

write_byte( 'S', 00H );    {Reserved}
write_byte( 'S', 00H );
write_byte( 'S', 00H );
write_byte( 'S', 00H );

write_byte( 'S', 00H );    {ABSTXT Address}
write_byte( 'S', 00H );
write_byte( 'S', 00H );

write_byte( 'S', 00H );    {ABSTXT Length - MUST BE DONE LAST}
write_byte( 'S', 00H );

num_abs_xbar_sets_output := 0;

write_byte( 'X', 0A2H ); { A2H = boot loadable module }

write_byte( 'X', 00H ); {total space}
write_byte( 'X', 00H );
write_byte( 'X', 00H );
write_byte( 'X', 00H );

for i := 0 to 7 do
  write_byte( 'X', ord(date_time.date[i]) );

for i := 0 to 7 do
  write_byte( 'X', ord(date_time.time[i]) );

write_byte( 'X', ord( 'i' ) );      {Creator}
write_byte( 'X', ord( 'R' ) );
write_byte( 'X', ord( 'M' ) );
write_byte( 'X', ord( 'X' ) );
write_byte( 'X', ord( ' ' ) );
write_byte( 'X', ord( 'P' ) );
write_byte( 'X', ord( 'F' ) );
write_byte( 'X', ord( 'P' ) );
```

```
write_byte( 'X', ord( ' ' ) );
write_byte( 'X', ord( 'C' ) );
write_byte( 'X', ord( 'r' ) );
write_byte( 'X', ord( 'o' ) );
write_byte( 'X', ord( 's' ) );
write_byte( 'X', ord( 's' ) );
write_byte( 'X', ord( 'b' ) );
write_byte( 'X', ord( 'a' ) );
write_byte( 'X', ord( 'r' ) );
write_byte( 'X', ord( '/' ) );
write_byte( 'X', ord( 'S' ) );
write_byte( 'X', ord( 'e' ) );
write_byte( 'X', ord( 'q' ) );
write_byte( 'X', ord( 'u' ) );
write_byte( 'X', ord( 'e' ) );
write_byte( 'X', ord( 'n' ) );
write_byte( 'X', ord( 'c' ) );
write_byte( 'X', ord( 'e' ) );
write_byte( 'X', ord( 'r' ) );
write_byte( 'X', ord( ' ' ) );
write_byte( 'X', ord( 'C' ) );
write_byte( 'X', ord( 'o' ) );
write_byte( 'X', ord( 'm' ) );
write_byte( 'X', ord( 'p' ) );
write_byte( 'X', ord( 'i' ) );
write_byte( 'X', ord( 'l' ) );
write_byte( 'X', ord( 'e' ) );
write_byte( 'X', ord( 'r' ) );
write_byte( 'X', ord( ' ' ) );
write_byte( 'X', ord( 'V' ) );
write_byte( 'X', ord( '1' ) );
write_byte( 'X', ord( '.' ) );
write_byte( 'X', ord( '0' ) );

write_byte( 'X', 00H );  {GDT}
write_byte( 'X', 00H );
write_byte( 'X', 00H );
write_byte( 'X', 00H );
write_byte( 'X', 00H );
write_byte( 'X', 00H );

write_byte( 'X', 00H );  {IDT}
write_byte( 'X', 00H );
write_byte( 'X', 00H );
write_byte( 'X', 00H );
write_byte( 'X', 00H );
write_byte( 'X', 00H );

write_byte( 'X', 00H );  {TSS}
write_byte( 'X', 00H );
```

```
    write_byte( 'X', 96 );      {ABSTXT Location}
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );

    write_byte( 'X', 00H );     {DEBTXT Location}
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );

    write_byte( 'X', 00H );     {ENDTXT Location - MUST BE DONE LAST}
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );

    write_byte( 'X', 00H );     {Next Partition}
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );

    write_byte( 'X', 00H );     {Reserved}
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );

    write_byte( 'X', 00H );     {ABSTXT Address}
    write_byte( 'X', 00H );
    write_byte( 'X', 00H );

    write_byte( 'X', 00H ); {ABSTXT Length - MUST BE DONE LAST }
    write_byte( 'X', 00H );

end;  { PROCEDURE initialize_output_files }


PROCEDURE finish_output_files;

const
        endtxt_offset = 84;
        length_offset = 99;

type
        temporary_type =
        record
                case byte of
                0: (b: array[0..3] of byte;);
                1: (i: integer;);
                2: (li: longint;);
        end;

var
```

```
        temporary: temporary_type;

begin
        temporary.li := position(seq_abs);
        write_byte('S',0FFH);  {checksum}

        seekwrite(seq_abs,endtxt_offset);
        write_byte('S',temporary.b[0]);
        write_byte('S',temporary.b[1]);
        write_byte('S',temporary.b[2]);
        write_byte('S',temporary.b[3]);

        seekwrite(seq_abs,length_offset);
        temporary.i := num_abs_seq_sets_output * 16;
        write_byte('S',temporary.b[0]);
        write_byte('S',temporary.b[1]);

        temporary.li := position(xbar_abs);
        write_byte('X',0FFH);  {checksum}

        seekwrite(xbar_abs,endtxt_offset);
        write_byte('X',temporary.b[0]);
        write_byte('X',temporary.b[1]);
        write_byte('X',temporary.b[2]);
        write_byte('X',temporary.b[3]);

        seekwrite(xbar_abs,length_offset);
        temporary.i := num_abs_xbar_sets_output * 64;
        write_byte('X',temporary.b[0]);
        write_byte('X',temporary.b[1]);

end;  { PROCEDURE finish_output_files }


FUNCTION read_char : char;

var
    file_char : char;
    ascii     : integer;

begin

    eof_flag := false;
    read(input_file,file_char);
    if eof(input_file) then begin
        eof_flag := true;
        stop_proc := true;
    end
    else begin
        stop_proc := false;
        ascii := ord(file_char);
```

```
        if ( ascii >= start_lower ) then
            file_char := chr(ascii - convert_upper);
        read_char := file_char;
    end;

end;  { FUNCTION read_char }



FUNCTION convert_to_integer( count : integer;
                             storage : digit_store ) : integer;

var
    temp_integer,
    index,
    i             : integer;

begin

    temp_integer := 0;
    for i := 0 to (count - 1) do begin
        index := count - i;
        temp_integer := (ord(storage[index]) - convert_numb)*power(i,10)
                        + temp_integer;
    end;
    convert_to_integer := temp_integer;

end;  { FUNCTION convert_to_integer }



PROCEDURE read_numb( type_read : char; var file_numb : integer );

var
    test_char : char;
    count,
    index :     integer;
    storage :   digit_store;

begin

    case type_read of
        'R' : begin
                count := 0;
                repeat
                    read(input_file,test_char);
                    if eof(input_file) then
                        stop_proc := true
                    else
                        stop_proc := false;
                    if not stop_proc then
```

```
                if ((test_char in ['0'..'9']) and
                   ((count+1) <= max_digits)) then begin
                    storage[count+1] := test_char;
                    count := count + 1;
                end;
            until ( (test_char in [',',':',chr(space)]) or stop_proc );
            if count = 0 then stop_proc := true;
            if not stop_proc then
            begin
                file_numb := convert_to_integer( count, storage );
                if ( file_numb >= ( max_proc_numb + 1 ) ) then
                   file_numb := file_numb - ( max_proc_numb + 1 );
            end;
        end;
'S' : begin
            count := 0;
            repeat
                read(input_file,test_char);
                if eof(input_file) then
                    eof_flag := true
                else
                    eof_flag := false;

                if ((test_char in ['0'..'9']) and
                   ((count+1) <= max_digits)) then begin
                    storage[count+1] := test_char;
                    count := count + 1;
                end;
            until ( (test_char in ['.',';',chr(space)]) );
            if count = 0 then stop_proc := true;
            if not stop_proc then begin
                file_numb := convert_to_integer( count, storage );
                if ( file_numb >= ( max_proc_numb + 1 ) ) then
                   file_numb := file_numb - ( max_proc_numb + 1 );
                if test_char = '.' then begin
                    count := 0;
                    repeat
                        read(input_file,test_char);
                        if eof(input_file) then
                            eof_flag := true
                        else
                            eof_flag := false;
                        if ((test_char in ['0'..'9']) and
                           ((count+1) <= max_digits)) then begin
                            storage[count+1] := test_char;
                            count := count + 1;
                        end;
                    until ( (test_char in [';',chr(space)]) );
                    if count = 0 then
                        processor.repeat_factor[comm_count] := 1
                    else
```

```
                        processor.repeat_factor[comm_count] :=
                            convert_to_integer( count, storage );
                    end
                    else
                        processor.repeat_factor[comm_count] := 1;
                end
            end;
    end;

    if stop_proc then error_log[2] := 1

end;  { FUNCTION read_numb }



PROCEDURE comment;

var
    test_char : char;

begin

    repeat
        test_char := read_char;
    until ( (test_char = ']') or stop_proc );
    if stop_proc then error_log[3] := 1;

end;  { PROCEDURE comment }


PROCEDURE check_loop;

var
    test_char : char;

begin

    test_char := read_char;
    if test_char = 'O' then begin
        test_char := read_char;
        if test_char = 'O' then begin
            test_char := read_char;
            if test_char <> 'P' then
                stop_proc := true;
        end
        else
            stop_proc := true;
    end
    else
        stop_proc := true;
```

```
    if stop_proc then error_log[5] := 1;

end;  { PROCEDURE check_loop }


PROCEDURE check_cycle;

var
    test_char : char;

begin

    test_char := read_char;
    if test_char = 'Y' then begin
        test_char := read_char;
        if test_char = 'C' then begin
            test_char := read_char;
            if test_char = 'L' then begin
                test_char := read_char;
                if test_char <> 'E' then
                    stop_proc := true;
            end
            else
                stop_proc := true;
        end
        else
            stop_proc := true;
    end
    else
        stop_proc := true;

    if not stop_proc then begin
        repeat
            test_char := read_char;
            if test_char = '[' then comment;
        until( (test_char = 'P') or stop_proc );
        if stop_proc then error_log[30] := 1;
    end
    else
        error_log[6] := 1;

end;  { PROCEDURE check_cycle }


PROCEDURE set_clear_data;

var
    i, j, k : integer;

begin
```

```
    for i := 0 to max_x_bus do
        for j := min_y_bus to max_proc_numb do
            clear_matrix[i][j] := 11;

    clear_tran.count := 0;
    for j := 1 to max_comm do begin
        clear_proc.repeat_factor[j] := 0;
        clear_tran.sender[j] := 0;
        clear_tran.receiver[j] := 0;
    end;
    for i := 0 to max_proc_numb do begin
        for j := 1 to max_comm do begin
            clear_proc.receiver[j][i] := 0;
            clear_proc.sender[j][i] := 0;
        end;
    end;
    for i := 1 to max_errors do
        error_log[i] := 0;

end;  { PROCEDURE set_clear_data }


PROCEDURE get_loop_number( var last_char : char );

var
    test_char : char;

begin

    check_loop;
    if not stop_proc then begin
        jump_number := step_number + 1;
        jump_cycle := cycle_number + 1;
        jump_flag := true;
        repeat
            test_char := read_char;
            if test_char = '[' then comment;
        until ( (test_char = 'C') or stop_proc );
        last_char := test_char;
        if stop_proc then error_log[8] := 1;
    end;

end;  { PROCEDURE get_loop_number }


PROCEDURE get_receivers( var found : boolean );

var
    test_char : char;
    stop_loop : boolean;
```

```
    index :     integer;

begin

    found := false;
    stop_loop := false;
    repeat

        read_numb( 'R', index );
        if stop_proc then begin
            stop_proc := true;
            error_log[9] := 1;
        end
        else begin
            found := true;
            if processor.receiver[comm_count][index] = 1 then begin
                stop_proc := true;
                error_log[10] := 1;
            end
            else
                processor.receiver[comm_count][index] := 1;
        end;
        if not stop_proc then
            repeat
                test_char := read_char;
                if test_char = '[' then comment;
                if stop_proc then error_log[11] := 1;
            until( (test_char in ['P','=','C']) or stop_proc );

        if not stop_proc then
            case test_char of
                '=' : begin
                         stop_loop := true;
                         if not found then stop_proc := true;
                      end;
                'C' : begin
                          stop_proc := true;
                          if stop_proc then error_log[13] := 1;
                      end;
            end;

    until( stop_loop or stop_proc );

end;  { PROCEDURE get_receivers }


PROCEDURE get_sender( var found : boolean; var input_char : char );

var
    test_char : char;
    index :     integer;
```

```
begin

    found := false;
    repeat
        test_char := read_char;
        if test_char = '[' then comment;
        if stop_proc then error_log[14] := 1;
    until( (test_char in ['P',';','C']) or stop_proc );

    if not stop_proc then begin
        case test_char of
            'P' : begin
                    read_numb( 'S', index );
                    if stop_proc then begin
                        stop_proc := true;
                        error_log[15]:= 1;
                    end
                    else begin
                        if processor.sender[comm_count][index] = 1 then begin
                            stop_proc := true;
                            error_log[16] := 1;
                        end
                        else
                            processor.sender[comm_count][index] := 1;
                        found := true;
                    end;
                end;
            ';' : begin
                    stop_proc := true;
                    error_log[17] := 1;
                end;
            'C' : begin
                    stop_proc := true;
                    error_log[18] := 1;
                end;
        end;
    end;

    if not stop_proc then begin
        if not eof_flag then
            repeat
                test_char := read_char;
                if test_char = '[' then comment;
            until( (test_char in ['P','C','L']) or stop_proc );
        if eof_flag then begin
            stop_proc := false;
            test_char := 'Q';
        end;
        input_char := test_char;
    end;
```

```
end;  { PROCEDURE get_sender }


PROCEDURE initialize_comm_info;

begin

    processor := clear_proc;
    xtox := clear_tran;
    xtoy := clear_tran;
    ytox := clear_tran;
    ytoy := clear_tran;

end;  { PROCEDURE initialize_comm_info }


PROCEDURE check_sender_receivers;

var
    i, j : integer;

begin

    for i := 0 to max_proc_numb do
        xbar_bus[i] := ' ';
    for j := 1 to comm_count do
        for i := 0 to max_proc_numb do begin
            if processor.sender[j][i] = 1 then
                if (xbar_bus[i] = 'R') or (xbar_bus[i] = 'S') then
                    stop_proc := true
                else
                    xbar_bus[i] := 'S';
            if processor.receiver[j][i] = 1 then
                if (xbar_bus[i] = 'R') or (xbar_bus[i] = 'S') then
                    stop_proc := true
                else
                    xbar_bus[i] := 'R';
        end;
    if stop_proc then error_log[25] := 1;

end;  { PROCEDURE check_sender_receivers }


PROCEDURE write_seq_to_output_file( last_one : boolean );

var
    i, j,
    out_count  : integer;
```

```
    first_comm : boolean;

begin

    first_comm := true;
    out_count := 0;
    writeln(setup,'    Cycle ',cycle_number);
    for j := 1 to comm_count do begin
        for i := 0 to max_proc_numb do
            if processor.receiver[j][i] = 1 then
                if first_comm then begin
                    write(setup,'        p',i);
                    first_comm := false;
                end
                else begin
                    write(setup,', p',i);
                    out_count := out_count + 1;
                    if out_count >= 7 then begin
                        writeln(setup,',');
                        first_comm := true;
                        out_count := 0;
                    end;
                end;

        write(setup,' := ');

        for i := 0 to max_proc_numb do
            if processor.sender[j][i] = 1 then
                write(setup,'p',i,'.',processor.repeat_factor[j],';');
        writeln(setup);
        first_comm := true;
    end;
    if last_one then begin
        writeln(setup);
        if jump_flag then
            writeln(setup,'    Loop to Cycle - ',jump_cycle)
        else
            writeln(setup,'    No loop');
    end;

end;  { PROCEDURE write_seq_to_output_file }


PROCEDURE build_data_word( now : integer; input_word : proc_array;
                           var output_word : word_array );

var
    i, j : integer;

begin
```

```
    for i := 0 to max_proc_numb do
        output_word[i] := 0;
    for j := 1 to comm_count do
        if now <= processor.repeat_factor[j] then
            for i := 0 to max_proc_numb do
                if input_word[j][i] = 1 then
                    output_word[i] := 1;

end;  { PROCEDURE build_data_word }


FUNCTION make_hex_numb( index : integer; input_word : word_array ) : integer;

var
    j, input_value : integer;

begin

    input_value := 0;
    for j := 1 to (bits_in_word div 2) do
       input_value := input_value + input_word[j+index] * power((j-1),2);
    make_hex_numb := input_value;

end;  { FUNCTION make_hex_numb }


PROCEDURE write_data_word( which_one : char; data_word : word_array );

var
    j, output_value : integer;

begin

    j := -1;
    repeat
        output_value := make_hex_numb( j, data_word );
        write_byte( which_one, output_value );
        j := j + 8;
    until( j > 23 );

end;  { PROCEDURE write_data_word }


PROCEDURE make_four_hex_numb( which_one : char; address : integer );

var
    upper, lower : integer;

begin
```

```
    upper := address div 256;
    lower := address mod 256;
    write_byte( which_one, lower );
    write_byte( which_one, upper );

end;  { PROCEDURE make_four_hex_numb }



PROCEDURE make_check_sum( which_one : char );

begin

    check_sum := - check_sum;
    write_byte( which_one, check_sum );

end;  { PROCEDURE make_check_sum }



PROCEDURE write_in_hex ( address_to_output : integer );

var
    output_number,
    left_over,
    hex_count :    word;

begin

    left_over := address_to_output;
    for hex_count := 3 downto 0 do begin
        output_number := left_over div power(hex_count, 16);
        left_over := left_over - output_number * power(hex_count, 16);
        if output_number < 10 then
            write(addr_setup,output_number:1)
        else if output_number = 10 then
            write(addr_setup,'A')
        else if output_number = 11 then
            write(addr_setup,'B')
        else if output_number = 12 then
            write(addr_setup,'C')
        else if output_number = 13 then
            write(addr_setup,'D')
        else if output_number = 14 then
            write(addr_setup,'E')
        else if output_number = 15 then
            write(addr_setup,'F');
    end;

end; { PROCEDURE write_in_hex }
```

```
PROCEDURE generate_seq_code( last_one : boolean );

var
    i, j,
    start :      integer;
    data_word :  word_array;

begin

    start := processor.repeat_factor[1];
    for i := 1 to comm_count do
        if processor.repeat_factor[i] > start then
            start := processor.repeat_factor[i];

    if gen_addr then write(addr_setup,'    ',cycle_number:4);
    for i := 1 to start do begin
        step_number := step_number + 1;
        if step_number <= max_step then begin

            num_abs_seq_sets_output := num_abs_seq_sets_output+1;

            build_data_word( i, processor.receiver, data_word );
            write_data_word( 'S', data_word );
            build_data_word( i, processor.sender, data_word );
            write_data_word( 'S', data_word );

            if i <> start then begin
                make_four_hex_numb( 'S', cycle_number - 1 );
                if gen_addr then begin
                    if i = 1 then begin
                        write(addr_setup,'           ');
                        write_in_hex(cycle_number - 1);
                    end
                    else begin
                        write(addr_setup,'                    ');
                        write_in_hex(cycle_number - 1);
                    end
                end
            end
            else if ( i = start) and last_one then begin
                if not jump_flag then
                    jump_cycle := cycle_number + 1;
                make_four_hex_numb( 'S', jump_cycle - 1 );
                if gen_addr then begin
                    if i = 1 then begin
                        write(addr_setup,'           ');
                        write_in_hex(jump_cycle - 1);
                    end
                    else begin
```

```
                    write(addr_setup,'                    ');
                    write_in_hex(jump_cycle - 1);
                end
            end
        end
        else begin
            make_four_hex_numb( 'S', cycle_number );;
            if gen_addr then begin
                if i = 1 then begin
                    write(addr_setup,'           ');
                    write_in_hex(cycle_number);
                end
                else begin
                    write(addr_setup,'                    ');
                    write_in_hex(cycle_number);
                end
            end
        end;
        if ( i = start) and last_one then begin
            if not jump_flag then
                jump_number := step_number + 1;
            make_four_hex_numb( 'S', jump_number - 1 );
            if gen_addr then begin
                write(addr_setup,'                    ');
                write_in_hex(jump_number - 1);
                writeln(addr_setup);
            end
        end
        else begin
            make_four_hex_numb( 'S', step_number );
            if gen_addr then begin
                write(addr_setup,'                    ');
                write_in_hex(step_number);
                writeln(addr_setup);
            end;
        end;
        write_byte('S',00H);  { Four unused memory locations }
        write_byte('S',00H);
        write_byte('S',00H);
        write_byte('S',00H);
    end;
end;

if step_number > max_step then begin
    stop_proc := true;
    error_log[28] := 1;
end
else begin
    writeln('  Cycle ',cycle_number);
    if gen_setup then
        write_seq_to_output_file( last_one );
```

```
    end;

end;  { PROCEDURE generate_seq_code }



PROCEDURE write_xbar_to_output_file;

var
    i, j : integer;

begin

    writeln(setup);
    writeln(setup,' Crossbar Setup');
    writeln(setup);
    write(setup,'    X');
    for i := 0 to max_x_bus do
        if i < 10 then
            write(setup,'    ',i)
        else
            write(setup,'   ',i);
    writeln(setup);
    write(setup,' Y     ');
    for i := 0 to max_x_bus do
        if xbar_bus[i] = ' ' then
            write(setup,'     ')
        else if xbar_bus[i] = 'R' then
            write(setup,' REC')
        else if xbar_bus[i] = 'S' then
            write(setup,' SND')
        else if xbar_bus[i] = 'H' then
            write(setup,' HLF');
    writeln(setup);
    writeln(setup);
    for i := min_y_bus to max_proc_numb do begin
        if (i - min_y_bus) < 10 then
            write(setup,' ',(i - min_y_bus))
        else
            write(setup,(i - min_y_bus));
        write(setup,' ');
        if xbar_bus[i] = ' ' then
            write(setup,'      ')
        else if xbar_bus[i] = 'R' then
            write(setup,'REC  ')
        else if xbar_bus[i] = 'S' then
            write(setup,'SND  ')
        else if xbar_bus[i] = 'H' then
            write(setup,'HLF  ');
        for j := 0 to max_x_bus do
            if xbar_matrix[j][i] = 11 then
```

```
            write(setup,'-    ')
        else if xbar_matrix[j][i] = 00 then
            write(setup,'YX   ')
        else if xbar_matrix[j][i] = 01 then
            write(setup,'XY   ');
      writeln(setup);
   end;
   writeln(setup);

end;  { PROCEDURE write_xbar_to_output_file }



PROCEDURE make_x_y_y_x( bus_to_bus : transfer );

var
    i, sender_index : integer;

begin

    i := 0;
    repeat
        i := i + 1;
        sender_index := bus_to_bus.sender[i];
        if sender_index in [0..15] then
            if xbar_matrix[sender_index]
                          [bus_to_bus.receiver[i]] <> 11 then
                stop_proc := true
            else
                xbar_matrix[sender_index]
                           [bus_to_bus.receiver[i]] := 01
        else if sender_index in [16..31] then
            if xbar_matrix[bus_to_bus.receiver[i]]
                          [sender_index] <> 11 then
                stop_proc := true
            else
                xbar_matrix[bus_to_bus.receiver[i]]
                           [sender_index] := 00;
    until ( (i = bus_to_bus.count) or stop_proc );
    if stop_proc then error_log[26] := 1;

end;  { PROCEDURE make_x_y_y_x }



PROCEDURE make_x_x_y_y( bus_to_bus : transfer );

var
    i, j, sender_index,
    receiver_index,
    empty_bus :      integer;
```

```
begin

    i := 0;
    repeat
        i := i + 1;
        sender_index := bus_to_bus.sender[i];
        receiver_index := bus_to_bus.receiver[i];
        if sender_index in [0..15] then begin
            empty_bus := -1;
            j := max_x_bus;
            repeat
                j := j + 1;
                if xbar_matrix[sender_index][j] = 01 then
                    empty_bus := j;
            until( (empty_bus <> -1) or (j = max_proc_numb) );
            if empty_bus = -1 then begin
                j := max_x_bus;
                repeat
                    j := j + 1;
                    if xbar_bus[j] = ' ' then
                        empty_bus := j;
                until ( (empty_bus <> -1) or (j = max_proc_numb) );
            end;
            if empty_bus = -1 then begin
                stop_proc := true;
                error_log[27] := 1;
            end
            else begin
                if xbar_bus[empty_bus] = ' ' then
                    xbar_bus[empty_bus] := 'H';
                if xbar_matrix[sender_index][empty_bus] <> 01 then
                    xbar_matrix[sender_index][empty_bus] := 01;
                if xbar_matrix[receiver_index][empty_bus] <> 11 then
                    stop_proc := true
                else
                    xbar_matrix[receiver_index][empty_bus] := 00;
                if stop_proc then error_log[27] := 1;
            end;
        end
        else if sender_index in [16..31] then begin
            empty_bus := -1;
            j := 0;
            repeat
                if xbar_matrix[j][sender_index] = 00 then
                    empty_bus := j;
                j := j + 1;
            until( (empty_bus <> -1) or (j > max_x_bus) );
            if empty_bus = -1 then begin
                j := 0;
                repeat
```

```
                if xbar_bus[j] = ' ' then
                    empty_bus := j;
                j := j + 1;
            until ( (empty_bus <> -1) or (j > max_x_bus) );
        end;
        if empty_bus = -1 then begin
            stop_proc := true;
            error_log[27] := 1;
        end
        else begin
            if xbar_bus[empty_bus] = ' ' then
                xbar_bus[empty_bus] := 'H';
            if xbar_matrix[empty_bus][sender_index] <> 00 then
                xbar_matrix[empty_bus][sender_index] := 00;
            if xbar_matrix[empty_bus][receiver_index] <> 11 then
                stop_proc := true
            else
                xbar_matrix[empty_bus][receiver_index] := 01;
            if stop_proc then error_log[27] := 1;
        end;
    end;
  until ( (i = bus_to_bus.count) or stop_proc );

end;  { PROCEDURE make_x_x_y_y }



PROCEDURE make_xbar_matrix;

var
    i, j : integer;

begin

    xbar_matrix := clear_matrix;

    if xtoy.count <> 0 then make_x_y_y_x( xtoy );
    if not stop_proc then begin
        if ytox.count <> 0 then make_x_y_y_x( ytox );
        if not stop_proc then begin
            if xtox.count <> 0 then make_x_x_y_y( xtox );
            if not stop_proc then
                if ytoy.count <> 0 then make_x_x_y_y( ytoy );
        end;
    end;

end;  { PROCEDURE make_xbar_matrix }



PROCEDURE rotate_matrix( rotate_count, x_index, y_index : integer );
```

```
var
    new_x,
    new_y,
    r_count,
    x_count,
    y_count,
    temp_data   : integer;
    temp_matrix : mat_array;

begin

    for r_count := 1 to rotate_count do begin
        temp_matrix := xbar_matrix;
        for x_count := x_index to (x_index + fourth_xbar) do
            for y_count := y_index to (y_index + fourth_xbar) do begin
                new_y := (x_count - x_index) + y_index;
                new_x := (fourth_xbar + x_index) - (y_count - y_index);
                temp_data := temp_matrix[x_count][y_count];
                if temp_data = 00 then
                    temp_data := 01
                else if temp_data = 01 then
                    temp_data := 00
                else
                    temp_data := 11;
                xbar_matrix[new_x][new_y] := temp_data;
            end;
    end;

end;  { PROCEDURE rotate_matrix }



PROCEDURE transform_xbar_matrix;

var
    x_index,
    y_index : integer;

begin

    x_index := 0;
    y_index := min_y_bus + fourth_xbar + 1;
    rotate_matrix( 1, x_index, y_index );

    x_index := fourth_xbar + 1;
    y_index := min_y_bus + fourth_xbar + 1;
    rotate_matrix( 2, x_index, y_index );

    x_index := fourth_xbar + 1;
    y_index := min_y_bus;
```

```
    rotate_matrix( 3, x_index, y_index );

end;  { PROCEDURE transform_xbar_matrix }



PROCEDURE generate_xbar_absolute_code( x_index, y_index : integer );

var
    index,
    x_count,
    y_count   : integer;
    data_word : word_array;

begin

    index := 0;
    for x_count := x_index to (x_index + fourth_xbar) do
        for y_count := y_index to (y_index + fourth_xbar) do begin
            if xbar_matrix[x_count][y_count] = 00 then begin
                data_word[index] := 0;
                data_word[index+1] := 0;
                index := index + 2;
            end
            else if xbar_matrix[x_count][y_count] = 01 then begin
                data_word[index] := 1;
                data_word[index+1] := 0;
                index := index + 2;
            end
            else begin
                data_word[index] := 1;
                data_word[index+1] := 1;
                index := index + 2;
            end;
            if index > 30 then begin
                index := 0;
                write_data_word( 'X', data_word );
            end;
        end;

end;  { PROCEDURE generate_xbar_absolute_code }



PROCEDURE write_xbar_data;

var
    x_index,
    y_index : integer;

begin
```

```
    x_index := 0;
    y_index := min_y_bus;
    generate_xbar_absolute_code( x_index, y_index );

    x_index := 0;
    y_index := min_y_bus + fourth_xbar + 1;
    generate_xbar_absolute_code( x_index, y_index );

    x_index := fourth_xbar + 1;
    y_index := min_y_bus + fourth_xbar + 1;
    generate_xbar_absolute_code( x_index, y_index );

    x_index := fourth_xbar + 1;
    y_index := min_y_bus;
    generate_xbar_absolute_code( x_index, y_index );

end;  { PROCEDURE write_xbar_data }



PROCEDURE complete_xbar_code;

var
    byte_count : integer;

begin

    transform_xbar_matrix;
    num_abs_xbar_sets_output := num_abs_xbar_sets_output+1;
    write_xbar_data;

end;  { PROCEDURE complete_xbar_code }



PROCEDURE generate_xbar_code;

var
    i, j,
    sender_index,
    receiver_index : integer;
    stop_loop      : boolean;

begin

    xtox.count := 0;
    xtoy.count := 0;
    ytox.count := 0;
    ytoy.count := 0;
    for j := 1 to comm_count do begin
```

```
        sender_index := 0;
        stop_loop := false;
        repeat
            if processor.sender[j][sender_index] = 1 then
                stop_loop := true
            else
                sender_index := sender_index + 1;
        until( stop_loop );
        for i := 0 to max_proc_numb do begin
            if (processor.receiver[j][i] = 1) and
               (i <> sender_index) then begin
                receiver_index := i;
                if sender_index in [0..15] then
                    if receiver_index in [0..15] then begin
                        xtox.count := xtox.count + 1;
                        xtox.sender[xtox.count] := sender_index;
                        xtox.receiver[xtox.count] := i;
                    end
                    else begin
                        xtoy.count := xtoy.count + 1;
                        xtoy.sender[xtoy.count] := sender_index;
                        xtoy.receiver[xtoy.count] := i;
                    end
                else if sender_index in [16..31] then
                    if receiver_index in [0..15] then begin
                        ytox.count := ytox.count + 1;
                        ytox.sender[ytox.count] := sender_index;
                        ytox.receiver[ytox.count] := i;
                    end
                    else begin
                        ytoy.count := ytoy.count + 1;
                        ytoy.sender[ytoy.count] := sender_index;
                        ytoy.receiver[ytoy.count] := i;
                    end;
            end;
        end;
    end;

    make_xbar_matrix;
    if not stop_proc then begin
        if gen_setup then
            write_xbar_to_output_file;
        complete_xbar_code;
    end;

end;  { PROCEDURE generate_xbar_code }
```

```
PROCEDURE process_cycle( var last_char : char );

var
    receivers_found,
    sender_found      : boolean;

begin

    check_cycle;

    if not stop_proc then begin
        cycle_number := cycle_number + 1;
        initialize_comm_info;
        comm_count := 0;
        repeat
            comm_count := comm_count + 1;
            if comm_count > max_comm then begin
                stop_proc := true;
                error_log[21] := 1;
            end;
            if not stop_proc then begin
                get_receivers( receivers_found );
                if receivers_found and not stop_proc then begin
                    get_sender( sender_found, last_char );
                    if not sender_found and not stop_proc then begin
                        stop_proc := true;
                        error_log[23] := 1;
                    end
                end
                else begin
                    stop_proc := true;
                    error_log[22] := 1;
                end;
            end;
        until( (last_char in ['C','Q','L']) or stop_proc );

        if not stop_proc then begin
            check_sender_receivers;
            if not stop_proc then begin
                if last_char = 'Q' then
                    generate_seq_code( true )
                else
                    generate_seq_code( false);
                if not stop_proc then generate_xbar_code;
            end;
        end;
    end;

end;  { PROCEDURE process_cycle }
```

```
PROCEDURE process_loop( var last_char : char );

begin

    get_loop_number( last_char );

end;  { PROCEDURE process_loop }



PROCEDURE process_file;

var
    test_char,
    last_char : char;

begin

    jump_flag := false;
    cycle_number := 0;
    step_number := 0;
    jump_number := 0;

    test_char := read_char;
    repeat
             if test_char = 'C' then process_cycle( last_char )
        else if test_char = 'Q' then stop_proc := true
        else if test_char = 'L' then process_loop( last_char )
        else if test_char = '{' then begin
                                    comment;
                                    last_char := read_char;
                               end
        else                last_char := read_char;
        test_char := last_char;
    until ( stop_proc );
    if cycle_number = 0 then error_log[24] := 1;

end;  { PROCEDURE process_file }



PROCEDURE error_check;

var
    i :          integer;
    no_errors :  boolean;
    error_file : text;

begin
```

```
no_errors := true;
for i := 1 to max_errors do
    if error_log[i] = 1 then no_errors := false;
if not no_errors then begin
    writeln;
    writeln(' Error detected in cycle - ',cycle_number,
            ' ,check error.dat file.');
    writeln;
    rewrite(error_file,'error.dat');
    writeln(error_file);
    writeln(error_file);
    writeln(error_file,'For cycle - ',cycle_number);
    writeln(error_file);
    for i := 1 to max_errors do
        if error_log[i] <> 0 then
        case i of
    1 : writeln(error_file,'  encountered eof while trying to read character');
    2 : writeln(error_file,'  encountered eof while trying to read number');
    3 : writeln(error_file,'  never found ] for comment end');
    4 : writeln(error_file,'  error checking BEGIN');
    5 : writeln(error_file,'  error checking LOOP');
    6 : writeln(error_file,'  error checking CYCLE');
    7 : writeln(error_file,'  error checking END');
    8 : writeln(error_file,'  never found CYCLE after LOOP');
    9 : writeln(error_file,'  receiver - no processor number found');
   10 : writeln(error_file,'  receiver - same receiver used again');
   11 : writeln(error_file,'  receiver - never found P, : or C');
   12 : writeln(error_file,'  receiver - := not found');
   13 : writeln(error_file,'  receiver - found C before any processors');
   14 : writeln(error_file,'  sender - never found P, ; or C');
   15 : writeln(error_file,'  sender - no processor number found');
   16 : writeln(error_file,'  sender - same sender used again');
   17 : writeln(error_file,'  sender - found ; before any processor');
   18 : writeln(error_file,'  sender - found C before any processor');
   19 : writeln(error_file,'  sender - never found P, C, L or E');
   20 : writeln(error_file,'  sender and receiver with same processor number');
   21 : writeln(error_file,'  over sixteen communications in one cycle');
   22 : writeln(error_file,'  no receivers found');
   23 : writeln(error_file,'  no senders found');
   24 : writeln(error_file,'  never found C or L to start processing');
   25 : writeln(error_file,'  processor used more than once in a cycle');
   26 : writeln(error_file,'  x to y or y to x bus conflict on xbar');
   27 : writeln(error_file,'  x to x or y to y no empty buses found');
   28 : writeln(error_file,'  exceded maximum step count for sequencer');
   29 : writeln(error_file,'  exceded maximum cycle count for sequencer');
   30 : writeln(error_file,'  after cycle never found P');
        end;
end;

end;  { PROCEDURE error_check }
```

```
{ Main Program }

begin

    open_files;
    initialize_output_files;
    set_clear_data;
    process_file;
    finish_output_files;
    error_check;

end.
```

E. Spinning Missile source code


```
File: BLOCK00.PAS

Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var    time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 16;

var
    low_last,
    high_last   : integer;

    rmf_table   : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );
```

```
    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;      index_table[2]  := 0.07;
    index_table[3]  := 0.25;     index_table[4]  := 0.35;
    index_table[5]  := 0.5;      index_table[6]  := 1.0;
    index_table[7]  := 1.47;     index_table[8]  := 1.5;
    index_table[9]  := 2.0;      index_table[10] := 2.5;
    index_table[11] := 3.0;      index_table[12] := 3.5;
    index_table[13] := 4.0;      index_table[14] := 4.393;
    index_table[15] := 4.394;    index_table[16] := 9.9995;

    rmf_table[1]  := 1.1184E-2;    rmf_table[2]  := 1.1236E-2;
    rmf_table[3]  := 1.1371E-2;    rmf_table[4]  := 1.14458E-2;
    rmf_table[5]  := 1.1558E-2;    rmf_table[6]  := 1.202E-2;
    rmf_table[7]  := 1.2494E-2;    rmf_table[8]  := 1.2524E-2;
    rmf_table[9]  := 1.3068E-2;    rmf_table[10] := 1.366E-2;
    rmf_table[11] := 1.4302E-2;    rmf_table[12] := 1.5008E-2;
    rmf_table[13] := 1.579E-2;     rmf_table[14] := 1.6464E-2;
    rmf_table[15] := 1.6466E-2;    rmf_table[16] := 1.6466E-2;

    for count := 2 to num_points do
        diff_table[count] := (rmf_table[count] - rmf_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }
```

```
Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow    : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
```

```
    Table RMF - Block 0
}
var
    sub_index,
    index       : integer;
    rmf         : real;

begin

    index := search_table( time );
    sub_index := index - 1;
    rmf := rmf_table[sub_index] +
           (diff_table[index] *
           (time - index_table[sub_index]));
    Send_Real_32bit( rmf );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK01.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 16;

var
    low_last,
    high_last   : integer;

    tf_table    : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;     index_table[2]  := 0.07;
    index_table[3]  := 0.25;    index_table[4]  := 0.35;
    index_table[5]  := 0.5;     index_table[6]  := 1.0;
    index_table[7]  := 1.47;    index_table[8]  := 1.5;
    index_table[9]  := 2.0;     index_table[10] := 2.5;
    index_table[11] := 3.0;     index_table[12] := 3.5;
    index_table[13] := 4.0;     index_table[14] := 4.393;
    index_table[15] := 4.394;   index_table[16] := 9.9995;

    tf_table[1]  := 3.3385E4;    tf_table[2]  := 3.5891E4;
```

```
    tf_table[3]  := 4.23325E4;   tf_table[4]  := 4.59115E4;
    tf_table[5]  := 5.128E4;     tf_table[6]  := 5.179E4;
    tf_table[7]  := 5.1527E4;    tf_table[8]  := 5.151E4;
    tf_table[9]  := 5.1025E4;    tf_table[10] := 5.0305E4;
    tf_table[11] := 5.001E4;     tf_table[12] := 5.0305E4;
    tf_table[13] := 5.0515E4;    tf_table[14] := 5.065E4;
    tf_table[15] := 0.0;         tf_table[16] := 0.0;

    for count := 2 to num_points do
        diff_table[count] := (tf_table[count] - tf_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
```

```
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table TF - Block 1
}
var
    sub_index,
    index      : integer;
    tf         : real;

begin

    index := search_table( time );
    sub_index := index - 1;
    tf := tf_table[sub_index] +
          (diff_table[index] *
          (time - index_table[sub_index]));
```

```
    Send_Real_32bit( tf );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK02.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 16;

var
    low_last,
    high_last   : integer;

    riyf_table  : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;     index_table[2]  := 0.07;
    index_table[3]  := 0.25;    index_table[4]  := 0.35;
    index_table[5]  := 0.5;     index_table[6]  := 1.0;
    index_table[7]  := 1.47;    index_table[8]  := 1.5;
    index_table[9]  := 2.0;     index_table[10] := 2.5;
    index_table[11] := 3.0;     index_table[12] := 3.5;
    index_table[13] := 4.0;     index_table[14] := 4.393;
    index_table[15] := 4.394;   index_table[16] := 9.9995;

    riyf_table[1]  := 49.05E-5;    riyf_table[2]  := 49.089E-5;
```

```
    riyf_table[3]  := 49.19E-5;      riyf_table[4]  := 49.246E-5;
    riyf_table[5]  := 49.33E-5;      riyf_table[6]  := 49.66E-5;
    riyf_table[7]  := 50.031E-5;     riyf_table[8]  := 50.055E-5;
    riyf_table[9]  := 50.45E-5;      riyf_table[10] := 50.805E-5;
    riyf_table[11] := 51.16E-5;      riyf_table[12] := 51.97E-5;
    riyf_table[13] := 52.78E-5;      riyf_table[14] := 53.45E-5;
    riyf_table[15] := 53.45E-5;      riyf_table[16] := 53.45E-5;

    for count := 2 to num_points do
        diff_table[count] := (riyf_table[count] - riyf_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
```

```
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table RIYF - Block 2
}
var
    sub_index,
    index       : integer;
    riyf        : real;

begin

    index := search_table( time );
    sub_index := index - 1;
    riyf := riyf_table[sub_index] +
            (diff_table[index] *
            (time - index_table[sub_index]));
```

```
    Send_Real_32bit( riyf );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK03.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 16;

var
    low_last,
    high_last    : integer;

    lclcgf_table : array [1..num_points] of real;
    index_table  : array [1..num_points] of real;
    diff_table   : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;     index_table[2]  := 0.07;
    index_table[3]  := 0.25;    index_table[4]  := 0.35;
    index_table[5]  := 0.5;     index_table[6]  := 1.0;
    index_table[7]  := 1.47;    index_table[8]  := 1.5;
    index_table[9]  := 2.0;     index_table[10] := 2.5;
    index_table[11] := 3.0;     index_table[12] := 3.5;
    index_table[13] := 4.0;     index_table[14] := 4.393;
    index_table[15] := 4.394;   index_table[16] := 9.9995;

    lclcgf_table[1]  := 6.75;      lclcgf_table[2]  := 6.7495;
```

```
    lclcgf_table[3]  := 6.7483;    lclcgf_table[4]  := 6.7476;
    lclcgf_table[5]  := 6.7465;    lclcgf_table[6]  := 6.743;
    lclcgf_table[7]  := 6.7651;    lclcgf_table[8]  := 6.7665;
    lclcgf_table[9]  := 6.79;      lclcgf_table[10] := 6.8405;
    lclcgf_table[11] := 6.891;     lclcgf_table[12] := 6.9945;
    lclcgf_table[13] := 7.098;     lclcgf_table[14] := 7.18;
    lclcgf_table[15] := 7.18;      lclcgf_table[16] := 7.18;

    for count := 2 to num_points do
        diff_table[count] := (lclcgf_table[count] - lclcgf_table(count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
```

```
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table LCLCGF - Block 3
}
var
    sub_index,
    index       : integer;
    lclcgf      : real;

begin

    index := search_table( time );
    sub_index := index - 1;
    lclcgf := lclcgf_table[sub_index] +
              (diff_table[index] *
              (time - index_table[sub_index]));
```

```
    Send_Real_32bit( lclcgf );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK04.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 16;

var
    low_last,
    high_last   : integer;

    ltf_table   : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;     index_table[2]  := 0.07;
    index_table[3]  := 0.25;    index_table[4]  := 0.35;
    index_table[5]  := 0.5;     index_table[6]  := 1.0;
    index_table[7]  := 1.47;    index_table[8]  := 1.5;
    index_table[9]  := 2.0;     index_table[10] := 2.5;
    index_table[11] := 3.0;     index_table[12] := 3.5;
    index_table[13] := 4.0;     index_table[14] := 4.393;
    index_table[15] := 4.394;   index_table[16] := 9.9995;

    ltf_table[1]  := 193.4;    ltf_table[2]  := 220.0 ;
```

```
    ltf_table[3]  := 260.0;    ltf_table[4]  := 260.0;
    ltf_table[5]  := 237.5;    ltf_table[6]  := 160.42;
    ltf_table[7]  := 80.0;     ltf_table[8]  := 0.0;
    ltf_table[9]  := 0.0;      ltf_table[10] := 0.0;
    ltf_table[11] := 0.0;      ltf_table[12] := 0.0;
    ltf_table[13] := 0.0;      ltf_table[14] := 0.0;
    ltf_table[15] := 0.0;      ltf_table[16] := 0.0;

    for count := 2 to num_points do
        diff_table[count] := (ltf_table[count] - ltf_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
```

```
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table LTF - Block 4
}
var
    sub_index,
    index       : integer;
    ltf         : real;

begin

    index := search_table( time );
    sub_index := index - 1;
    ltf := ltf_table[sub_index] +
           (diff_table[index] *
           (time - index_table[sub_index]));
```

```
    Send_Real_32bit( ltf );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK05.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 16;

var
    low_last,
    high_last   : integer;

    acd0f_table : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;     index_table[2]  := 0.8;
    index_table[3]  := 1.2;     index_table[4]  := 1.5;
    index_table[5]  := 1.6;     index_table[6]  := 1.7;
    index_table[7]  := 1.8;     index_table[8]  := 2.0;
    index_table[9]  := 2.1;     index_table[10] := 2.3;
    index_table[11] := 2.5;     index_table[12] := 2.7;
    index_table[13] := 3.0;     index_table[14] := 3.5;
    index_table[15] := 4.394;   index_table[16] := 9.9995;

    acd0f_table[1]  := 9.625E-2;     acd0f_table[2]  := 7.0E-2;
```

```
    acd0f_table[3]  := 6.235E-2;     acd0f_table[4]  := 5.63E-2;
    acd0f_table[5]  := 6.185E-2;     acd0f_table[6]  := 7.72E-2;
    acd0f_table[7]  := 9.25E-2;      acd0f_table[8]  := 1.307E-1;
    acd0f_table[9]  := 1.4155E-1;    acd0f_table[10] := 1.3725E-1;
    acd0f_table[11] := 1.274E-1;     acd0f_table[12] := 1.201E-1;
    acd0f_table[13] := 1.0915E-1;    acd0f_table[14] := 9.785E-2;
    acd0f_table[15] := 8.335E-2;     acd0f_table[16] := 8.335E-2;

    for count := 2 to num_points do
        diff_table[count] := (acd0f_table[count] - acd0f_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
```

```
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table ACD0F - Block 5
}
var
    sub_index,
    index      : integer;
    acd0f      : real;

begin

    index := search_table( time );
    sub_index := index - 1;
    acd0f := acd0f_table[sub_index] +
             (diff_table[index] *
             (time - index_table[sub_index]));
```

```
    Send_Real_32bit( acd0f );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK06.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 16;

var
    low_last,
    high_last   : integer;

    cmaf_table  : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;     index_table[2]  := 0.8;
    index_table[3]  := 1.2;     index_table[4]  := 1.5;
    index_table[5]  := 1.6;     index_table[6]  := 1.7;
    index_table[7]  := 1.8;     index_table[8]  := 2.0;
    index_table[9]  := 2.1;     index_table[10] := 2.3;
    index_table[11] := 2.5;     index_table[12] := 2.7;
    index_table[13] := 3.0;     index_table[14] := 3.5;
    index_table[15] := 4.394;   index_table[16] := 9.9995;

    cmaf_table[1]  := -7.5925;    cmaf_table[2]  := -9.8925;
```

```
    cmaf_table[3]  := -8.9775;    cmaf_table[4]  := -6.2;
    cmaf_table[5]  := -4.8225;    cmaf_table[6]  := -3.805;
    cmaf_table[7]  := -4.235;     cmaf_table[8]  := -6.355;
    cmaf_table[9]  := -7.48;      cmaf_table[10] := -8.655;
    cmaf_table[11] := -8.385;     cmaf_table[12] := -7.715;
    cmaf_table[13] := -6.21;      cmaf_table[14] := -3.945;
    cmaf_table[15] := -1.655;     cmaf_table[16] := -1.655;

    for count := 2 to num_points do
        diff_table[count] := (cmaf_table[count] - cmaf_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
```

```
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table CMAF - Block 6
}
var
    sub_index,
    index       : integer;
    cmaf        : real;

begin

    index := search_table( time );
    sub_index := index - 1;
    cmaf := cmaf_table[sub_index] +
            (diff_table[index] *
            (time - index_table[sub_index]));
```

```
    Send_Real_32bit( cmaf );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK07.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 16;

var
    low_last,
    high_last   : integer;

    cmqf_table  : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;     index_table[2]  := 0.8;
    index_table[3]  := 1.2;     index_table[4]  := 1.5;
    index_table[5]  := 1.6;     index_table[6]  := 1.7;
    index_table[7]  := 1.8;     index_table[8]  := 2.0;
    index_table[9]  := 2.1;     index_table[10] := 2.3;
    index_table[11] := 2.5;     index_table[12] := 2.7;
    index_table[13] := 3.0;     index_table[14] := 3.5;
    index_table[15] := 4.394;   index_table[16] := 9.9995;

    cmqf_table[1]  := -1054.4;    cmqf_table[2]  := -1094.0;
```

```
    cmqf_table[3]  := -1114.0;    cmqf_table[4]  := -1060.0;
    cmqf_table[5]  := -1035.2;    cmqf_table[6]  := -1010.8;
    cmqf_table[7]  := -986.0;     cmqf_table[8]  := -936.8;
    cmqf_table[9]  := -938.8;     cmqf_table[10] := -940.8;
    cmqf_table[11] := -960.0;     cmqf_table[12] := -981.2;
    cmqf_table[13] := -1013.2;    cmqf_table[14] := -972.4;
    cmqf_table[15] := -878.8;     cmqf_table[16] := -878.8;

    for count := 2 to num_points do
        diff_table[count] := (cmqf_table[count] - cmqf_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
```

```
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table CMQF - Block 7
}
var
    sub_index,
    index      : integer;
    cmqf       : real;

begin

    index := search_table( time );
    sub_index := index - 1;
    cmqf := cmqf_table[sub_index] +
            (diff_table[index] *
            (time - index_table[sub_index]));
```

```
   Send_Real_32bit( cmqf );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK08.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 8;
    z0         = 3990.0;

var
    low_last,
    high_last   : integer;

    wnsf_table  : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    low_last := 1;
    high_last := num_points;

    index_table[1] := 0.0;       index_table[2] := 2.0E3;
    index_table[3] := 3.7E3;     index_table[4] := 4.0E3;
    index_table[5] := 6.0E3;     index_table[6] := 8.0E3;
    index_table[7] := 10.0E3;    index_table[8] := 12.0E3;

    wnsf_table[1]  := 1.175;       wnsf_table[2]  := 0.53986;
    wnsf_table[3]  := 0.0;         wnsf_table[4]  := -0.09524;
    wnsf_table[5]  := -0.73016;    wnsf_table[6]  := -1.36508;
    wnsf_table[7]  := -2.0;        wnsf_table[8]  := -2.63492;
```

```
    for count := 2 to num_points do
        diff_table[count] := (wnsf_table[count] - wnsf_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
              else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
```

```
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table WNSF - Block 8
}
var
    sub_index,
    index      : integer;
    z,
    zprime,
    wnsf       : real;

begin

    Receive_Real_32bit( z );
    zprime := z + z0;
    index := search_table( zprime );
    sub_index := index - 1;
    wnsf := wnsf_table[sub_index] +
            (diff_table[index] *
            (zprime - index_table[sub_index]));
    Send_Real_32bit( wnsf );

end;. { Procedure Evaluate_table }
```

```
File: BLOCK09.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 8;
    z0         = 3990.0;

var
    low_last,
    high_last   : integer;

    wwef_table  : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    low_last := 1;
    high_last := num_points;

    index_table[1] := 0.0;       index_table[2] := 2.0E3;
    index_table[3] := 3.7E3;     index_table[4] := 4.0E3;
    index_table[5] := 6.0E3;     index_table[6] := 8.0E3;
    index_table[7] := 10.0E3;    index_table[8] := 12.0E3;

    wwef_table[1]  := -2.0;    wwef_table[2]  := -2.0;
    wwef_table[3]  := -2.0;    wwef_table[4]  := -2.0;
    wwef_table[5]  := -2.0;    wwef_table[6]  := -2.0;
    wwef_table[7]  := -2.0;    wwef_table[8]  := -2.0;
```

```
    for count := 2 to num_points do
        diff_table[count] := (wwef_table[count] - wwef_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
```

```
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table WWEF - Block 9
}
var
    sub_index,
    index       : integer;
    z,
    zprime,
    wwef        : real;

begin

    Receive_Real_32bit( z );
    zprime := z + z0;
    index := search_table( zprime );
    sub_index := index - 1;
    wwef := wwef_table[sub_index] +
            (diff_table[index] *
            (zprime - index_table[sub_index]));
    Send_Real_32bit( wwef );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK10.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 8;
    z0         = 3990.0;

var
    low_last,
    high_last   : integer;

    rhof_table  : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    low_last := 1;
    high_last := num_points;

    index_table[1] := 0.0;      index_table[2] := 2.0E3;
    index_table[3] := 3.7E3;    index_table[4] := 4.0E3;
    index_table[5] := 6.0E3;    index_table[6] := 8.0E3;
    index_table[7] := 10.0E3;   index_table[8] := 12.0E3;

    rhof_table[1]  := 2.1163E-3;    rhof_table[2]  := 2.0696E-3;
    rhof_table[3]  := 1.9846E-3;    rhof_table[4]  := 1.9696E-3;
    rhof_table[5]  := 1.8673E-3;    rhof_table[6]  := 1.7684E-3;
    rhof_table[7]  := 1.6751E-3;    rhof_table[8]  := 1.585E-3;
```

```
    for count := 2 to num_points do
        diff_table[count] := (rhof_table[count] - rhof_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }


Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }


Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
       (search_value < index_table[ihigh]) then begin
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
```

```
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table RHOF - Block 10
}
var
    sub_index,
    index       : integer;
    z,
    zprime,
    rhof        : real;

begin

    Receive_Real_32bit( z );
    zprime := z + z0;
    index := search_table( zprime );
    sub_index := index - 1;
    rhof := rhof_table[sub_index] +
            (diff_table[index] *
            (zprime - index_table[sub_index]));
    Send_Real_32bit( rhof );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK11.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 13;
    asound     = 1117.4;

var
    low_last,
    high_last   : integer;
    inv_asound  : real;

    acnaf_table : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    inv_asound := 1.0 / asound;
    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;      index_table[2]  := 0.235;
    index_table[3]  := 0.5;      index_table[4]  := 0.609;
    index_table[5]  := 0.777;    index_table[6]  := 1.005;
    index_table[7]  := 1.119;    index_table[8]  := 1.235;
    index_table[9]  := 1.41;     index_table[10] := 1.772;
    index_table[11] := 2.025;    index_table[12] := 2.494;
    index_table[13] := 2.56;
```

```
    acnaf_table[1]  := 27.852;    acnaf_table[2]  := 30.228;
    acnaf_table[3]  := 31.926;    acnaf_table[4]  := 31.728;
    acnaf_table[5]  := 29.238;    acnaf_table[6]  := 26.304;
    acnaf_table[7]  := 28.206;    acnaf_table[8]  := 29.028;
    acnaf_table[9]  := 27.672;    acnaf_table[10] := 25.29;
    acnaf_table[11] := 24.078;    acnaf_table[12] := 22.608;
    acnaf_table[13] := 22.488;

    for count := 2 to num_points do
        diff_table[count] := (acnaf_table[count] - acnaf_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
```

```
      (search_value < index_table[ihigh]) then begin
       pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table ACNAF - Block 11
}
var
    sub_index,
    index      : integer;
    mach,
    us,
    acnaf      : real;

begin

    Receive_Real_32bit( us );
    mach := us * inv_asound;
```

```
    index := search_table( mach );
    sub_index := index - 1;
    acnaf := acnaf_table[sub_index] +
             (diff_table[index] *
             (mach - index_table[sub_index]));
    Send_Real_32bit( acnaf );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK12.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 13;
    asound     = 1117.4;

var
    low_last,
    high_last   : integer;
    inv_asound  : real;

    cldtf_table : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    inv_asound := 1.0 / asound;
    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;     index_table[2]  := 0.235;
    index_table[3]  := 0.5;     index_table[4]  := 0.609;
    index_table[5]  := 0.777;   index_table[6]  := 1.005;
    index_table[7]  := 1.119;   index_table[8]  := 1.235;
    index_table[9]  := 1.41;    index_table[10] := 1.772;
    index_table[11] := 2.025;   index_table[12] := 2.494;
    index_table[13] := 2.56;
```

```
    cldtf_table[1]  := 4.103;     cldtf_table[2]  := 3.764;
    cldtf_table[3]  := 3.337;     cldtf_table[4]  := 3.2571;
    cldtf_table[5]  := 3.134;     cldtf_table[6]  := 3.228;
    cldtf_table[7]  := 3.3162;    cldtf_table[8]  := 3.4058;
    cldtf_table[9]  := 3.198;     cldtf_table[10] := 2.44;
    cldtf_table[11] := 2.02;      cldtf_table[12] := 1.4135;
    cldtf_table[13] := 1.337;

    for count := 2 to num_points do
        diff_table[count] := (cldtf_table[count] - cldtf_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
             else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
```

```
      (search_value < index_table[ihigh]) then begin
       pivot( ihigh, ilow, search_value );

  end
  else if search_value = index_table[ilow] then begin
      ihigh := ilow + 1;
  end
  else if search_value = index_table[ihigh] then begin
      ilow := ihigh;
      ihigh := ilow + 1;
  end
  else if search_value < index_table[1] then begin
      ihigh := 2;
      ilow := 1;
  end
  else if search_value > index_table[num_points] then begin
      ihigh := num_points;
      ilow := ihigh - 1;
  end
  else if search_value > index_table[ihigh] then begin
      ihigh := num_points;
      pivot( ihigh, ilow, search_value );
  end
  else if search_value < index_table[ilow] then begin
      ilow := 1;
      pivot( ihigh, ilow, search_value );
  end;
  low_last := ilow;
  high_last := ihigh;

  search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table CLDTF - Block 12
}
var
    sub_index,
    index      : integer;
    mach,
    us,
    cldtf      : real;

begin

    Receive_Real_32bit( us );
    mach := us * inv_asound;
```

```
    index := search_table( mach );
    sub_index := index - 1;
    cldtf := cldtf_table[sub_index] +
             (diff_table[index] *
             (mach - index_table[sub_index]));
    Send_Real_32bit( cldtf );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK13.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var time, integration_step : Real;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    num_points = 13;
    asound     = 1117.4;

var
    low_last,
    high_last   : integer;
    inv_asound  : real;

    clpf_table  : array [1..num_points] of real;
    index_table : array [1..num_points] of real;
    diff_table  : array [1..num_points] of real;



Procedure Initialize_table;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, integration_step, message_size );

    inv_asound := 1.0 / asound;
    low_last := 1;
    high_last := num_points;

    index_table[1]  := 0.0;      index_table[2]  := 0.235;
    index_table[3]  := 0.5;      index_table[4]  := 0.609;
    index_table[5]  := 0.777;    index_table[6]  := 1.005;
    index_table[7]  := 1.119;    index_table[8]  := 1.235;
    index_table[9]  := 1.41;     index_table[10] := 1.772;
    index_table[11] := 2.025;    index_table[12] := 2.494;
    index_table[13] := 2.56;
```

```
    clpf_table[1]  := 8.716;      clpf_table[2]  := 8.7254;
    clpf_table[3]  := 8.726;      clpf_table[4]  := 8.7476;
    clpf_table[5]  := 8.781;      clpf_table[6]  := 11.335;
    clpf_table[7]  := 10.4;       clpf_table[8]  := 9.5532;
    clpf_table[9]  := 8.2825;     clpf_table[10] := 8.042;
    clpf_table[11] := 7.826;      clpf_table[12] := 7.5253;
    clpf_table[13] := 7.483;

    for count := 2 to num_points do
        diff_table[count] := (clpf_table[count] - clpf_table[count-1]) /
                             (index_table[count] - index_table[count-1]);

end;  { Procedure Initialize_table }



Procedure pivot( var ihigh, ilow : integer; search_value : real );

var
    ipiv : integer;

begin

    while ((ihigh - ilow) > 1) do begin
        ipiv := (ihigh + ilow) div 2;
        if search_value = index_table[ipiv] then begin
            ilow := ipiv;
            ihigh := ilow + 1;
        end
        else if search_value < index_table[ipiv] then
                ihigh := ipiv
              else
                ilow := ipiv;
    end;

end;  { Procedure pivot }



Function search_table( search_value : real ) : integer;

var
    ihigh,
    ilow   : integer;

begin

    ilow := low_last;
    ihigh := high_last;
    if (search_value > index_table[ilow]) and
```

```
      (search_value < index_table[ihigh]) then begin
        pivot( ihigh, ilow, search_value );

    end
    else if search_value = index_table[ilow] then begin
        ihigh := ilow + 1;
    end
    else if search_value = index_table[ihigh] then begin
        ilow := ihigh;
        ihigh := ilow + 1;
    end
    else if search_value < index_table[1] then begin
        ihigh := 2;
        ilow := 1;
    end
    else if search_value > index_table[num_points] then begin
        ihigh := num_points;
        ilow := ihigh - 1;
    end
    else if search_value > index_table[ihigh] then begin
        ihigh := num_points;
        pivot( ihigh, ilow, search_value );
    end
    else if search_value < index_table[ilow] then begin
        ilow := 1;
        pivot( ihigh, ilow, search_value );
    end;
    low_last := ilow;
    high_last := ihigh;

    search_table := ihigh;

end;  { Function search_table }



Procedure Evaluate_table( time : real );
{
    Table CLPF - Block 13
}
var
    sub_index,
    index      : integer;
    mach,
    us,
    clpf      : real;

begin

    Receive_Real_32bit( us );
    mach := us * inv_asound;
```

```
    index := search_table( mach );
    sub_index := index - 1;
    clpf := clpf_table[sub_index] +
            (diff_table[index] *
            (mach - index_table[sub_index]));
    Send_Real_32bit( clpf );

end;.  { Procedure Evaluate_table }
```

```
File: BLOCK14.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    asound = 1117.4;
    g      = 32.17;
    us0    = 15.33;
    theta0 = 0.942;

var
    rm, t, acd0, rho,
    rs, wns, theta, vs,
    qs, ws, us,
    thetpr, sinth, costh,
    wx, rsvs, qsws, uswx,
    uswxsq, v13, v14,
    uforce, v16, usdot    : real;


Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    Y := us0;
    time := 0.0;

end;


Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
```

```
                                first_eval : boolean );
{
   Basic Integrator - State us - Block 14
}
begin

   us := Y;
   Send_Real_32bit( us );
   Receive_Real_32bit( vs );
   Receive_Real_32bit( theta );
   Receive_Real_32bit( rs );
   Receive_Real_32bit( qs );
   Receive_Real_32bit( t );
   Receive_Real_32bit( ws );
   Receive_Real_32bit( rho );
   Receive_Real_32bit( acd0 );
   Receive_Real_32bit( wns );
   Receive_Real_32bit( rm );
{
if first_eval then Output_Message( Real_32bit, us / asound, 1 );
}
   thetpr := theta + theta0;
   sinth := sin( thetpr );
   costh := cos( thetpr );
   wx := wns * costh;
   rsvs := rs * vs;
   qsws := qs * ws;
   uswx := us + wx;
   uswxsq := uswx * uswx;
   v13 := rho * uswxsq;
   v14 := acd0 * v13;
   uforce := t - 0.5 * v14;
   v16 := rm * uforce;

   usdot := v16 - g * sinth + rsvs - qsws;
   Y_dot := usdot;

end;.
```

File: BLOCK15.PAS

```
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                    first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    vs0 = 0.0;

var
    fy, phi, fz, rm,
    acna, rho, us, wwe,
    rs, vs,
    sinphi, cosphi, wy,
    rsus, vswy, rhous,
    v19, v20, fty,
    vforce, v22, vsdot  : real;


Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    Y := vs0;
    time := 0.0;

end;


Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                 first_eval : boolean );
{
    Basic Integrator - State vs - Block 15
```

```
)
begin

    vs := Y;
    Receive_Real_32bit( us );
    Send_Real_32bit( vs );
    Send_Real_32bit( vs );
    Receive_Real_32bit( rs );
    Receive_Real_32bit( phi );
    Receive_Real_32bit( wwe );
    Receive_Real_32bit( rho );
    Receive_Real_32bit( fz );
    Receive_Real_32bit( fy );
    Receive_Real_32bit( acna );
    Receive_Real_32bit( rm );

    sinphi := sin( phi );
    cosphi := cos( phi );
    wy := wwe;
    rsus := rs * us;
    vswy := vs - wy;
    rhous := rho * us;
    v19 := acna * rhous;
    v20 := v19 * vswy;
    fty := fy * cosphi + fz * sinphi;
    vforce := fty - 0.5 * v20;
    v22 := vforce * rm;
    vsdot := v22 - rsus;
    Y_dot := vsdot;

end;.
```

```
File: BLOCK16.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    g      = 32.17;
    ws0    = 0.0;
    theta0 = 0.942;

var
    qs, us, theta, rm,
    fz, phi, fy, wns,
    acna, rho, ws,
    thetpr, sinth, costh,
    sinphi, cosphi, wz,
    qsus, rhous, v19, ftz,
    wswz, v24, wforce,
    v26, wsdot             : real;


Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    Y := ws0;
    time := 0.0;

end;


Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
```

```
                    first_eval : boolean );
{
    Basic Integrator - State ws - Block 16
}
begin

    ws := Y;
    Receive_Real_32bit( us );
    Receive_Real_32bit( theta );
    Receive_Real_32bit( qs );
    Receive_Real_32bit( phi );
    Send_Real_32bit( ws );
    Receive_Real_32bit( rho );
    Receive_Real_32bit( fz );
    Receive_Real_32bit( fy );
    Receive_Real_32bit( wns );
    Receive_Real_32bit( acna );
    Receive_Real_32bit( rm );

    thetpr := theta + theta0;
    sinth := sin( thetpr );
    costh := cos( thetpr );
    sinphi := sin( phi );
    cosphi := cos( phi );
    wz := wns * sinth;
    qsus := qs * us;
    rhous := rho * us;
    v19 := acna * rhous;
    wswz := ws + wz;
    v24 := wswz * v19;
    ftz := fz * cosphi - fy * sinphi;
    wforce := -ftz - 0.5 * v24;
    v26 := rm * wforce;
    wsdot := qsus + g * costh + v26;
    Y_dot := wsdot;

end;.
```

```
File: BLOCK17.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    ps0 = 0.0;
    ix  = 25.77;
    dx  = -2.0;
    a   = 2.64;
    d   = 1.833;
    dt  = 0.05236;

var
    rho, clp, us,
    cldt, lt, ps,
    p7, p8, p10, p11,
    rhous, psclp,
    uscldt, v29,
    v30, psdot        : real;


Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    p7 := 1.0 / ix;
    p8 := dx / ix;
    p10 := a * d * d * 0.25;
    p11 := a * d * dt * 0.5;
    Y := ps0;
    time := 0.0;
```

```
end;


Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State ps - Block 17
}
begin

    ps := Y;
    Receive_Real_32bit( us );
    Send_Real_32bit( ps );
    Receive_Real_32bit( lt );
    Receive_Real_32bit( rho );
    Receive_Real_32bit( clp );
    Receive_Real_32bit( cldt );
{
if first_eval then Output_Message( Real_32bit, ps, 1 );
}
    rhous := rho * us;
    psclp := ps * clp;
    uscldt := us * cldt;
    v29 := -p10 * psclp + p11 * uscldt;
    v30 := rhous * v29;
    psdot := p7 * v30 + p7 * lt + p8 * ps;
    Y_dot := psdot;

end;.
```

```
File: BLOCK18.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    phi0   = 0.0;
    theta0 = 0.942;

var
    ps, theta, rs,
    phi,
    thetpr, sinth, costh,
    rcos, phidot          : real;


Procedure Initial_Conditions;

var
    message_type, message_size: integer;

begin

    input_message( message_type, h, message_size );

    Y := phi0;
    time := 0.0;

end;


Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State phi - Block 18
}
begin
```

```
    phi := Y;
    Receive_Real_32bit( ps );
    Receive_Real_32bit( theta );
    Receive_Real_32bit( rs );
    Send_Real_32bit( phi );
{
if first_eval then Output_Message( Real_32bit, phi, 1 );
}
    thetpr := theta + theta0;
    sinth := sin( thetpr );
    costh := cos( thetpr );
    rcos := 1.0 / costh;
    phidot := ps + rcos * sinth * rs;
    Y_dot := phidot;

end;.
```

```
File: BLOCK19.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    z0     = 3990.0;
    zic    = 0.0;
    theta0 = 0.942;

var
    us, theta, ws,
    z,
    thetpr, sinth, costh,
    zdot                 : real;


Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    Y := zic;
    time := 0.0;

end;



Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State z - Block 19
}
```

```
begin

    z := Y;
    Receive_Real_32bit( us );
    Send_Real_32bit( z );
    Receive_Real_32bit( theta );
    Receive_Real_32bit( ws );
{
if first_eval then Output_Message( Real_32bit, z + z0, 1 );
}
    thetpr := theta + theta0;
    sinth := sin( thetpr );
    costh := cos( thetpr );
    zdot := us * sinth - ws * costh;
    Y_dot := zdot;

end;.
```

File: BLOCK20.PAS

```
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    gt20 = 0.0;
    t1   = 0.873;
    s1   = 0.914;
    gain = 28.65;
    s2   = 0.246;
    tau2 = 1.875E-3;

var
    theta, phi, psi,
    gt2,
    p22, p23, p30,
    sinphi, cosphi,
    gammat, gt1,
    gt2dot                : real;


Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    p22 := s1 * gain;
    p23 := s2 * gain;
    p30 := 1.0 / tau2;
    Y := gt20;
    time := 0.0;

end;
```

```
Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State gt2 - Block 20
}
begin

    gt2 := Y;
    Send_Real_32bit( gt2 );
    Receive_Real_32bit( theta );
    Receive_Real_32bit( psi );
    Receive_Real_32bit( phi );

    sinphi := sin( phi );
    cosphi := cos( phi );
    gammat := theta * cosphi + psi * sinphi;
    if time < t1 then
        gt1 := gammat * p22
    else
        gt1 := gammat * p23;
    gt2dot := p30 * ( gt1 - gt2 );
    Y_dot := gt2dot;

end;.
```

```
File: BLOCK21.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    gp20 = 0.0;
    t1   = 0.873;
    s1   = 0.914;
    gain = 28.65;
    s2   = 0.246;
    tau2 = 1.875E-3;

var
    theta, phi, psi,
    gp2,
    p22, p23, p30,
    sinphi, cosphi,
    gammap, gp1,
    gp2dot               : real;


Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    p22 := s1 * gain;
    p23 := s2 * gain;
    p30 := 1.0 / tau2;
    Y := gp20;
    time := 0.0;

end;
```

```
Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State gp2 - Block 21
}
begin

    gp2 := Y;
    Send_Real_32bit( gp2 );
    Receive_Real_32bit( theta );
    Receive_Real_32bit( psi );
    Receive_Real_32bit( phi );

    sinphi := sin( phi );
    cosphi := cos( phi );
    gammap := psi * cosphi - theta * sinphi;
    if time < t1 then
        gp1 := gammap * p22
    else
        gp1 := gammap * p23;
    gp2dot := p30 * ( gp1 - gp2 );
    Y_dot := gp2dot;

end;.
```

```
File: BLOCK22.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    gt40 = 0.0;
    t1   = 0.873;
    gain = 28.65;
    s1   = 0.914;
    s2   = 0.246;
    tau1 = 17.34E-3;
    tau2 = 1.875E-3;
    tau3 = 670.0E-6;

var
    theta, phi, psi,
    gt2, gt4,
    p22, p23, p27,
    p28, p29,
    sinphi, cosphi,
    gammat, gt1, gt3,
    gt4dot                : real;


Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    p22 := s1 * gain;
    p23 := s2 * gain;
    p27 := tau1 / tau2;
    p28 := 1.0 - p27;
```

```
    p29 := 1.0 / tau3;
    Y := gt40;
    time := 0.0;

end;



Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State gt4 - Block 22
}
begin

    gt4 := Y;
    Receive_Real_32bit( gt2 );
    Send_Real_32bit( gt4 );
    Receive_Real_32bit( theta );
    Receive_Real_32bit( psi );
    Receive_Real_32bit( phi );

    sinphi := sin( phi );
    cosphi := cos( phi );
    gammat := theta * cosphi + psi * sinphi;
    if time < t1 then
        gt1 := gammat * p22
    else
        gt1 := gammat * p23;
    gt3 := p27 * gt1 + p28 * gt2;
    gt4dot := p29 * ( gt3 - gt4 );
    Y_dot := gt4dot;

end;.
```

```
File: BLOCK23.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    gp40 = 0.0;
    t1   = 0.873;
    gain = 28.65;
    s1   = 0.914;
    s2   = 0.246;
    tau1 = 17.34E-3;
    tau2 = 1.875E-3;
    tau3 = 670.0E-6;

var
    theta, phi, psi,
    gp2, gp4,
    p22, p23, p27,
    p28, p29,
    sinphi, cosphi,
    gammap, gp1, gp3,
    gp4dot                : real;


Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    p22 := s1 * gain;
    p23 := s2 * gain;
    p27 := tau1 / tau2;
    p28 := 1.0 - p27;
```

```
    p29 := 1.0 / tau3;
    Y := gp40;
    time := 0.0;

end;



Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State gp4 - Block 23
}
begin

    gp4 := Y;
    Receive_Real_32bit( gp2 );
    Send_Real_32bit( gp4 );
    Receive_Real_32bit( theta );
    Receive_Real_32bit( psi );
    Receive_Real_32bit( phi );

    sinphi := sin( phi );
    cosphi := cos( phi );
    gammap := psi * cosphi - theta * sinphi;
    if time < t1 then
        gp1 := gammap * p22
    else
        gp1 := gammap * p23;
    gp3 := p27 * gp1 + p28 * gp2;
    gp4dot := p29 * ( gp3 - gp4 );
    Y_dot := gp4dot;

end;.
```

```
File: BLOCK24.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    int_limit,
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    gt50  = 0.0;
    c     = 0.5E-6;
    r1    = 20.0E3;
    r2    = 147.0E3;
    v1    = 0.24;
    v2    = 0.18;
    eomax = 2.5;
    eimax = 0.5;

var
    gt4, gt5,
    p35, p36,
    rsw_minus, rsw_plus,
    gt6, gt5dot          : real;
    setff, resetff,
    gt6hi, gt6lo         : boolean;

    ly_srtff,
    oldy                 : array[1..2] of boolean;


Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );
```

```
    p35 := 1.0 / ( r1 * c );
    p36 := 1.0 / ( r2 * c );
    Y := gt50;
    int_limit := eimax;
    time := 0.0;

end;



Function srtff( index : integer; first_time : boolean ) : boolean;

begin

    if time = 0.0 then begin
        ly_srtff[index] := false;
        oldy[index] := false;
    end
    else if first_time then begin
        if setff then ly_srtff[index] := true;
        if resetff then ly_srtff[index] := false;
        if (not setff) and (not resetff) then ly_srtff[index] := oldy[index];
        if setff and resetff then ly_srtff[index] := not oldy[index];
        oldy[index] := ly_srtff[index];
    end;
    srtff := ly_srtff[index];

end;  { Function srtff }



Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State gt5 - Block 24
}
begin

    gt5 := Y;
    Send_Real_32bit( gt5 );
    Receive_Real_32bit( gt4 );
{
if first_eval then Output_Message( Real_32bit, gt5, 1 );
}
    if gt5 > v1 then setff := true else setff := false;
    if gt5 < -v2 then resetff := true else resetff := false;
    gt6hi := srtff( 1, first_eval );
    if gt5 < -v1 then setff := true else setff := false;
    if gt5 > v2 then resetff := true else resetff := false;
    gt6lo := srtff( 2, first_eval );
    if gt6hi then
```

```
        rsw_plus := eomax
    else
        rsw_plus := 0.0;
    if gt6lo then
        rsw_minus := eomax
    else
        rsw_minus := 0.0;
    gt6 := rsw_plus - rsw_minus;
    gt5dot := p35 * gt4 - p36 * gt6;
    Y_dot := gt5dot;

end;.
```

```
File: BLOCK25.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    int_limit,
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    gp50  = 0.0;
    c     = 0.5E-6;
    r1    = 20.0E3;
    r2    = 147.0E3;
    v1    = 0.24;
    v2    = 0.18;
    eomax = 2.5;
    eimax = 0.5;

var
    gp4, gp5,
    p35, p36,
    rsw_minus, rsw_plus,
    gp6, gp5dot          : real;
    setff, resetff,
    gp6hi, gp6lo         : boolean;

    ly_srtff,
    oldy                 : array [1..2] of boolean;



Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );
```

```
    p35 := 1.0 / ( r1 * c );
    p36 := 1.0 / ( r2 * c );
    Y := gp50;
    int_limit := eimax;
    time := 0.0;

end;



Function srtff( index : integer; first_time : boolean ) : boolean;

begin

    if time = 0.0 then begin
        ly_srtff[index] := false;
        oldy[index] := false;
    end
    else if first_time then begin
        if setff then ly_srtff[index] := true;
        if resetff then ly_srtff[index] := false;
        if (not setff) and (not resetff) then ly_srtff[index] := oldy[index];
        if setff and resetff then ly_srtff[index] := not oldy[index];
        oldy[index] := ly_srtff[index];
    end;
    srtff := ly_srtff[index];

end;  { Function srtff }



Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State gp5 - Block 25
}
begin

    gp5 := Y;
    Send_Real_32bit( gp5 );
    Receive_Real_32bit( gp4 );
{
if first_eval then Output_Message( Real_32bit, gp5, 1 );
}
    if gp5 > v1 then setff := true else setff := false;
    if gp5 < -v2 then resetff := true else resetff := false;
    gp6hi := srtff( 1, first_eval );
    if gp5 < -v1 then setff := true else setff := false;
    if gp5 > v2 then resetff := true else resetff := false;
    gp6lo := srtff( 2, first_eval );
    if gp6hi then
```

```
        rsw_plus := eomax
    else
        rsw_plus := 0.0;
    if gp6lo then
        rsw_minus := eomax
    else
        rsw_minus := 0.0;
    gp6 := rsw_plus - rsw_minus;
    gp5dot := p35 * gp4 - p36 * gp6;
    Y_dot := gp5dot;

end;.
```

```
File: BLOCK26.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean;
                                   var mode : boolean );

Public Solve_Differential_Equation;

Var
    int_limit,
    time, h, Y, Y_dot : real;
    first_eval, mode  : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    fzic  = 0.0;
    trf   = 4.0E-3;
    eomax = 2.5;
    v1    = 0.24;
    v2    = 0.18;
    tdel  = 8.0E-3;
    fside = 380.0;

var
    gt5, fz,
    half_eomax, p43,
    gt7,
    gt6, rsw_plus,
    rsw_minus,
    switch_up,
    switch_down,
    fzdot              : real;
    delay_head,
    delay_tail,
    delay_index        : integer;
    gt6hi, gt6lo,
    setff, resetff,
    fzpos, gt7hi,
    gt7lo, tdt0,
    tut0, tup, tdown   : boolean;

    ly_srtff,
    oldy               : array [1..4] of boolean;
    ly_dlyff,
    oldx               : array [1..2] of boolean;
```

```
    delay_gt6         : array [0..99] of real;

Procedure Initial_Conditions;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    half_eomax := eomax / 2.0;
    p43 := fside / trf;
    Y := fzic;
    int_limit := fside;
    delay_index := trunc(tdel/h) * 4;
    for count := 0 to (delay_index - 1) do
        delay_gt6[count] := 0.0;
    time := 0.0;
    delay_head := 0;
    delay_tail := delay_index - 1;

end;


Function srtff( index : integer; first_time : boolean ) : boolean;

begin

    if time = 0.0 then begin
        ly_srtff[index] := false;
        oldy[index] := false;
    end
    else if first_time then begin
        if setff then ly_srtff[index] := true;
        if resetff then ly_srtff[index] := false;
        if (not setff) and (not resetff) then ly_srtff[index] := oldy[index];
        if setff and resetff then ly_srtff[index] := not oldy[index];
        oldy[index] := ly_srtff[index];
    end;
    srtff := ly_srtff[index];

end;  { Function srtff }


Function dlyff( first_time, value : boolean; index : integer ) : boolean;

begin
```

```
    if time = 0.0 then begin
        ly_dlyff[index] := false;
        oldx[index] := value
    end
    else if first_time then begin
        ly_dlyff[index] := oldx[index];
        oldx[index] := value
    end;
    dlyff := ly_dlyff[index];

end;  { Function dlyff }



Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean;
                                var mode : boolean );
{
    Basic Integrator - State fz - Block 26
}
var
    count,
    kount : integer;

begin

    fz := Y;
    Receive_Real_32bit( gt5 );
    Send_Real_32bit( fz );
{
if first_eval then Output_Message( Real_32bit, fz, 1 );
}
    if gt5 > v1 then setff := true else setff := false;
    if gt5 < -v2 then resetff := true else resetff := false;
    gt6hi := srtff( 1, first_eval );
    if gt5 < -v1 then setff := true else setff := false;
    if gt5 > v2 then resetff := true else resetff := false;
    gt6lo := srtff( 2, first_eval );
    if gt6hi then
        rsw_plus := eomax
    else
        rsw_plus := 0.0;
    if gt6lo then
        rsw_minus := eomax
    else
        rsw_minus := 0.0;
    gt6 := rsw_plus - rsw_minus;

    gt7 := delay_gt6[delay_head];
    delay_gt6[delay_tail] := gt6;
```

```
    delay_head := delay_head + 1;
    delay_tail := delay_tail + 1;
    if delay_head = delay_index then delay_head := 0;
    if delay_tail = delay_index then delay_tail := 0;

    if fz > 0.0 then fzpos := true else fzpos := false;
    if gt7 > half_eomax then gt7hi := true else gt7hi := false;
    if gt7 < -half_eomax then gt7lo := true else gt7lo := false;
    setff := dlyff( first_eval, gt7hi, 1 ) and ( not gt7hi );
    resetff := not fzpos;
    tdt0 := srtff( 3, first_eval );
    setff := dlyff( first_eval, gt7lo, 2 ) and ( not gt7lo );
    resetff := fzpos;
    tut0 := srtff( 4, first_eval );
    tup := gt7hi or tut0;
    tdown := gt7lo or tdt0;
    if tup then switch_up := p43 else switch_up := 0.0;
    if tdown then switch_down := -p43 else switch_down := 0.0;
    fzdot := switch_up + switch_down;
    Y_dot := fzdot;
    if first_eval then mode := not( tup or tdown );

end;.
```

```
File: BLOCK27.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean;
                                   var mode : boolean );

Public Solve_Differential_Equation;

Var
    int_limit,
    time, h, Y, Y_dot : real;
    first_eval, mode  : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    fyic  = 0.0;
    trf   = 4.0E-3;
    eomax = 2.5;
    v1    = 0.24;
    v2    = 0.18;
    tdel  = 8.0E-3;
    fside = 380.0;

var
    gp5, fy,
    half_eomax, p43,
    gp7,
    gp6, rsw_plus,
    rsw_minus,
    switch_up,
    switch_down,
    fydot             : real;
    delay_head,
    delay_tail,
    delay_index       : integer;
    gp6hi, gp6lo,
    setff, resetff,
    fypos, gp7hi,
    gp7lo, pdt0,
    put0, pup, pdown  : boolean;

    ly_srtff,
    oldy              : array [1..4] of boolean;
    ly_dlyff,
    oldx              : array [1..2] of boolean;
```

```
    delay_gp6          : array [0..99] of real;

Procedure Initial_Conditions;

var
    count : integer;
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    half_eomax := eomax / 2.0;
    p43 := fside / trf;
    Y := fyic;
    int_limit := fside;
    delay_index := trunc(tdel/h) * 4;
    for count := 0 to (delay_index - 1) do
        delay_gp6[count] := 0.0;
    time := 0.0;
    delay_head := 0;
    delay_tail := delay_index - 1;

end;



Function srtff( index : integer; first_time : boolean ) : boolean;

begin

    if time = 0.0 then begin
        ly_srtff[index] := false;
        oldy[index] := false;
    end
    else if first_time then begin
        if setff then ly_srtff[index] := true;
        if resetff then ly_srtff[index] := false;
        if (not setff) and (not resetff) then ly_srtff[index] := oldy[index];
        if setff and resetff then ly_srtff[index] := not oldy[index];
        oldy[index] := ly_srtff[index];
    end;
    srtff := ly_srtff[index];

end;  { Function srtff }



Function dlyff( first_time, value : boolean; index : integer ) : boolean;

begin
```

```
    if time = 0.0 then begin
        ly_dlyff[index] := false;
        oldx[index] := value
    end
    else if first_time then begin
        ly_dlyff[index] := oldx[index];
        oldx[index] := value
    end;
    dlyff := ly_dlyff[index];

end;  { Function dlyff }



Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean;
                                var mode : boolean );
{
    Basic Integrator - State fy - Block 27
}
var
    count,
    kount : integer;

begin

    fy := Y;
    Receive_Real_32bit( gp5 );
    Send_Real_32bit( fy );
{
if first_eval then Output_Message( Real_32bit, fy, 1 );
}
    if gp5 > v1 then setff := true else setff := false;
    if gp5 < -v2 then resetff := true else resetff := false;
    gp6hi := srtff( 1, first_eval );
    if gp5 < -v1 then setff := true else setff := false;
    if gp5 > v2 then resetff := true else resetff := false;
    gp6lo := srtff( 2, first_eval );
    if gp6hi then
        rsw_plus := eomax
    else
        rsw_plus := 0.0;
    if gp6lo then
        rsw_minus := eomax
    else
        rsw_minus := 0.0;
    gp6 := rsw_plus - rsw_minus;

    gp7 := delay_gp6[delay_head];
    delay_gp6[delay_tail] := gp6;
```

```
    delay_head := delay_head + 1;
    delay_tail := delay_tail + 1;
    if delay_head = delay_index then delay_head := 0;
    if delay_tail = delay_index then delay_tail := 0;

    if fy > 0.0 then fypos := true else fypos := false;
    if gp7 > half_eomax then gp7hi := true else gp7hi := false;
    if gp7 < -half_eomax then gp7lo := true else gp7lo := false;
    setff := dlyff( first_eval, gp7hi, 1 ) and ( not gp7hi );
    resetff := not fypos;
    pdt0 := srtff( 3, first_eval );
    setff := dlyff( first_eval, gp7lo, 2 ) and ( not gp7lo );
    resetff := fypos;
    put0 := srtff( 4, first_eval );
    pup := gp7hi or put0;
    pdown := gp7lo or pdt0;
    if pup then switch_up := p43 else switch_up := 0.0;
    if pdown then switch_down := -p43 else switch_down := 0.0;
    fydot := switch_up + switch_down;
    Y_dot := fydot;
    if first_eval then mode := not( pup or pdown );

end;.
```

```
File: BLOCK28.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    qs0    = 0.0;
    a      = 2.64;
    d      = 1.833;
    ix     = 25.77;
    theta0 = 0.942;

var
    riy, rho, us, cmq,
    ws, wns, theta,
    cma, rs, ps, lcmlcg,
    fz, phi, fy, qs,
    p9, p10, ftz,
    sinphi, cosphi,
    thetpr, sinth, wz,
    rsps, rhous, wswz,
    v36, v33, v37,
    qsdot              : real;



Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    p9 := a * d * 0.5;
    p10 := a * d * d * 0.25;
    Y := qs0;
```

```
    time := 0.0;

end;



Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State qs - Block 28
}
begin

    qs := Y;
    Receive_Real_32bit( us );
    Receive_Real_32bit( ps );
    Receive_Real_32bit( theta );
    Receive_Real_32bit( rs );
    Send_Real_32bit( qs );
    Receive_Real_32bit( phi );
    Receive_Real_32bit( ws );
    Receive_Real_32bit( rho );
    Receive_Real_32bit( fz );
    Receive_Real_32bit( fy );
    Receive_Real_32bit( wns );
    Receive_Real_32bit( riy );
    Receive_Real_32bit( lcmlcg );
    Receive_Real_32bit( cmq );
    Receive_Real_32bit( cma );
{
if first_eval then Output_Message( Real_32bit, qs, 1 );
}
    sinphi := sin( phi );
    cosphi := cos( phi );
    ftz := fz * cosphi - fy * sinphi;
    v33 := lcmlcg * ftz;
    rhous := rho * us;
    thetpr := theta + theta0;
    sinth := sin( thetpr );
    wz := wns * sinth;
    wswz := ws + wz;
    rsps := rs * ps;
    v36 := rhous * ( p10 * qs * cmq + p9 * wswz * cma ) - ix * rsps;
    v37 := v36 - v33;
    qsdot := riy * v37;
    Y_dot := qsdot;

end;.
```

```
File: BLOCK29.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    thetic = 0.0;

var
    qs, theta, thetd : real;



Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    Y := thetic;
    time := 0.0;

end;



Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State theta - Block 29
}
begin

    theta := Y;
    Send_Real_32bit( theta );
```

```
    Receive_Real_32bit( qs );
{
if first_eval then Output_Message( Real_32bit, theta, 1 );
}
    thetd := qs;
    Y_dot := thetd;

end;.
```

```
File: BLOCK30.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    theta0 = 0.942;
    psiic  = 0.0;

var
    rs, theta, psi,
    thetpr, costh,
    rcos, psidot    : real;



Procedure Initial_Conditions;

var
    message_type, message_size : integer;

begin

    input_message( message_type, h, message_size );

    Y := psiic;
    time := 0.0;

end;



Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State psi - Block 30
}
begin
```

```
    psi := Y;
    Receive_Real_32bit( theta );
    Receive_Real_32bit( rs );
    Send_Real_32bit( psi );
{
if first_eval then Output_Message( Real_32bit, psi, 1 );
}
    thetpr := theta + theta0;
    costh := cos( thetpr );
    rcos := 1.0 / costh;
    psidot := rs * rcos;
    Y_dot := psidot

end;.
```

```
File: BLOCK31.PAS
Module Problem_Specifications;
Public Problem_Specifications;

   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

$Include (':PFP:include/target.pas')

Private Problem_Specifications;

const
    rs0    = 0.0;
    a      = 2.64;
    d      = 1.833;
    ix     = 25.77;

var
    riy, rho, us, cmq,
    vs, wwe,
    cma, ps, qs, lcmlcg,
    fz, phi, fy, rs,
    p9, p10, fty,
    sinphi, cosphi, wy,
    psqs, rhous, vswy,
    v40, v43, v44,
    rsdot                : real;



Procedure Initial_Conditions;

var
    message_type, message_size: integer;

begin

    input_message( message_type, h, message_size );

    p9 := a * d * 0.5;
    p10 := a * d * d * 0.25;
    Y := rs0;
    time := 0.0;
```

```
end;


Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                first_eval : boolean );
{
    Basic Integrator - State rs - Block 31
}
begin

    rs := Y;
    Receive_Real_32bit( us );
    Receive_Real_32bit( ps );
    Receive_Real_32bit( vs );
    Send_Real_32bit( rs );
    Receive_Real_32bit( qs );
    Receive_Real_32bit( phi );
    Receive_Real_32bit( wwe );
    Receive_Real_32bit( rho );
    Receive_Real_32bit( fz );
    Receive_Real_32bit( fy );
    Receive_Real_32bit( riy );
    Receive_Real_32bit( lcmlcg );
    Receive_Real_32bit( cmq );
    Receive_Real_32bit( cma );

if first_eval then
begin
        Output_Message( Character_08bit, 'time=', 5 );
        Output_Message( Real_32bit, time, 1 );
        Output_Message( Character_08bit, 'rs=', 3 );
        Output_Message( Real_32bit, rs, 1 );
        Output_Nl;
end;
    sinphi := sin( phi );
    cosphi := cos( phi );
    fty := fy * cosphi + fz * sinphi;
    v40 := lcmlcg * fty;

    rhous := rho * us;
    wy := wwe;
    vswy := vs - wy;
    psqs := ps * qs;
    v43 := rhous * ( p10 * rs * cmq - p9 * vswy * cma ) + ix * psqs;

    v44 := v43 - v40;
    rsdot := riy * v44;
    Y_dot := rsdot;

end;.
```

```
File: LMRK4.PAS
Module Solve_Differential_Equation;

Public Problem_Specifications;
   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean;
                                   var mode : boolean );

Public Solve_Differential_Equation;

Var
    int_limit,
    time, h, Y, Y_dot : real;
    first_eval, mode  : boolean;

Program Solve_Differential_Equation;



Procedure Integrate_with_Limit_and_Mode;
{
          Integration Routine using Fourth Order Runge-Kutta.
                                                                      }
var
    dummy_mode,
    upper_limit,
    in_limit      : boolean;
    inv_three,
    k1, k2, k3,
    half_h        : real;

begin

    inv_three := 1 / 3.0;
    half_h := 0.5 * h;
    in_limit := false;
    while true do begin
        Evaluate_Derivatives(Y_dot, Y, time, true, mode);
            if mode then begin
                k1 := 0.0;
                in_limit := false;
            end
            else begin
                if not in_limit then begin
                    if Y >= int_limit then begin
                        Y := int_limit;
                        in_limit := true;
                        upper_limit := true;
                    end
                    else if Y <= -int_limit then begin
```

```
                    Y := -int_limit;
                    in_limit := true;
                    upper_limit := false;
                end
            end
            else
                if upper_limit then begin
                    if Y_dot < 0.0 then in_limit := false
                end
                else begin
                    if Y_dot > 0.0 then in_limit := false
                end;
            if in_limit then Y_dot := 0.0;
            k1 := Y + half_h * Y_dot;
        end;
    Evaluate_Derivatives(Y_dot, k1, time + half_h, false, dummy_mode);
        if mode then
            k2 := 0.0
        else begin
            if in_limit then Y_dot := 0.0;
            k2 := Y + half_h * Y_dot;
        end;
    Evaluate_Derivatives(Y_dot, k2, time + half_h, false, dummy_mode);
        if mode then
            k3 := 0.0
        else begin
            if in_limit then Y_dot := 0.0;
            k3 := Y + h * Y_dot;
        end;
    Evaluate_Derivatives(Y_dot, k3, time + h, false, dummy_mode);
        if mode then
            Y := 0.0
        else begin
            if in_limit then Y_dot := 0.0;
            Y := ( k1 + 2.0*k2 + k3 + half_h * Y_dot - Y ) * inv_three;
        end;
    time := time + h;
  end;

end; { PROCEDURE Integrate_with_Limit_and_Mode }



begin

  Initial_Conditions;
  Integrate_with_Limit_and_Mode;

end.
```

```
File: LRK4.PAS
Module Solve_Differential_Equation;

Public Problem_Specifications;
   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    int_limit,
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

Program Solve_Differential_Equation;



Procedure Integrate_with_Limit;
{
          Integration Routine using Fourth Order Runge-Kutta.
                                                                }

var
    upper_limit,
    in_limit      : boolean;
    inv_three,
    k1, k2, k3,
    half_h        : real;

begin

    inv_three := 1 / 3.0;
    half_h := 0.5 * h;
    in_limit := false;
    while true do begin
        Evaluate_Derivatives(Y_dot, Y, time, true);
            if not in_limit then begin
                if Y >= int_limit then begin
                    Y := int_limit;
                    in_limit := true;
                    upper_limit := true;
                end
                else if Y <= -int_limit then begin
                    Y := -int_limit;
                    in_limit := true;
                    upper_limit := false;
                end
            end
            else
                if upper_limit then begin
```

```
                if Y_dot < 0.0 then in_limit := false
            end
            else begin
                if Y_dot > 0.0 then in_limit := false
            end;
        if in_limit then Y_dot := 0.0;
        k1 := Y + half_h * Y_dot;
    Evaluate_Derivatives(Y_dot, k1, time + half_h, false);
        if in_limit then Y_dot := 0.0;
        k2 := Y + half_h * Y_dot;
    Evaluate_Derivatives(Y_dot, k2, time + half_h, false);
        if in_limit then Y_dot := 0.0;
        k3 := Y + h * Y_dot;
    Evaluate_Derivatives(Y_dot, k3, time + h, false);
        if in_limit then Y_dot := 0.0;
        Y := ( k1 + 2.0*k2 + k3 + half_h * Y_dot - Y ) * inv_three;
    time := time + h;
  end;

end; { PROCEDURE Integrate_with_Limit }



begin

  Initial_Conditions;
  Integrate_with_Limit;

end.
```

```
File: MAKEFILE
PASFLAGS = large optimize(1) symbolspace(64)

PROGRAM = \
          block00.bl \
          block01.bl \
          block02.bl \
          block03.bl \
          block04.bl \
          block05.bl \
          block06.bl \
          block07.bl \
          block08.bl \
          block09.bl \
          block10.bl \
          block11.bl \
          block12.bl \
          block13.bl \
          block14.bl \
          block15.bl \
          block16.bl \
          block17.bl \
          block18.bl \
          block19.bl \
          block20.bl \
          block21.bl \
          block22.bl \
          block23.bl \
          block24.bl \
          block25.bl \
          block26.bl \
          block27.bl \
          block28.bl \
          block29.bl \
          block30.bl \
          block31.bl \
          crossbar.bl \
          sequencer.bl

default: $(PROGRAM)

block00.bl:         block00.obj table.obj
          submit :PFP:csd/pasbldl( block00, 'block00.obj,table.obj' )

block00.obj:        block00.pas
          pas286 block00.pas $(PASFLAGS)

block01.bl:         block01.obj table.obj
          submit :PFP:csd/pasbldl( block01, 'block01.obj,table.obj' )

block01.obj:        block01.pas
```

```
        pas286 block01.pas $(PASFLAGS)

block02.bl:         block02.obj table.obj
        submit :PFP:csd/pasbldl( block02, 'block02.obj,table.obj' )

block02.obj:        block02.pas
        pas286 block02.pas $(PASFLAGS)

block03.bl:         block03.obj table.obj
        submit :PFP:csd/pasbldl( block03, 'block03.obj,table.obj' )

block03.obj:        block03.pas
        pas286 block03.pas $(PASFLAGS)

block04.bl:         block04.obj table.obj
        submit :PFP:csd/pasbldl( block04, 'block04.obj,table.obj' )

block04.obj:        block04.pas
        pas286 block04.pas $(PASFLAGS)

block05.bl:         block05.obj table.obj
        submit :PFP:csd/pasbldl( block05, 'block05.obj,table.obj' )

block05.obj:        block05.pas
        pas286 block05.pas $(PASFLAGS)

block06.bl:         block06.obj table.obj
        submit :PFP:csd/pasbldl( block06, 'block06.obj,table.obj' )

block06.obj:        block06.pas
        pas286 block06.pas $(PASFLAGS)

block07.bl:         block07.obj table.obj
        submit :PFP:csd/pasbldl( block07, 'block07.obj,table.obj' )

block07.obj:        block07.pas
        pas286 block07.pas $(PASFLAGS)

block08.bl:         block08.obj table.obj
        submit :PFP:csd/pasbldl( block08, 'block08.obj,table.obj' )

block08.obj:        block08.pas
        pas286 block08.pas $(PASFLAGS)

block09.bl:         block09.obj table.obj
        submit :PFP:csd/pasbldl( block09, 'block09.obj,table.obj' )

block09.obj:        block09.pas
        pas286 block09.pas $(PASFLAGS)

block10.bl:         block10.obj table.obj
```

```
        submit :PFP:csd/pasbldl( block10, 'block10.obj,table.obj' )

block10.obj:        block10.pas
        pas286 block10.pas $(PASFLAGS)

block11.bl:         block11.obj table.obj
        submit :PFP:csd/pasbldl( block11, 'block11.obj,table.obj' )

block11.obj:        block11.pas
        pas286 block11.pas $(PASFLAGS)

block12.bl:         block12.obj table.obj
        submit :PFP:csd/pasbldl( block12, 'block12.obj,table.obj' )

block12.obj:        block12.pas
        pas286 block12.pas $(PASFLAGS)

block13.bl:         block13.obj table.obj
        submit :PFP:csd/pasbldl( block13, 'block13.obj,table.obj' )

block13.obj:        block13.pas
        pas286 block13.pas $(PASFLAGS)

block14.bl:         block14.obj rk4.obj
        submit :PFP:csd/pasbldl( block14, 'block14.obj,rk4.obj' )

block14.obj:        block14.pas
        pas286 block14.pas $(PASFLAGS)

block15.bl:         block15.obj rk4.obj
        submit :PFP:csd/pasbldl( block15, 'block15.obj,rk4.obj' )

block15.obj:        block15.pas
        pas286 block15.pas $(PASFLAGS)

block16.bl:         block16.obj rk4.obj
        submit :PFP:csd/pasbldl( block16, 'block16.obj,rk4.obj' )

block16.obj:        block16.pas
        pas286 block16.pas $(PASFLAGS)

block17.bl:         block17.obj rk4.obj
        submit :PFP:csd/pasbldl( block17, 'block17.obj,rk4.obj' )

block17.obj:        block17.pas
        pas286 block17.pas $(PASFLAGS)

block18.bl:         block18.obj rk4.obj
        submit :PFP:csd/pasbldl( block18, 'block18.obj,rk4.obj' )

block18.obj:        block18.pas
```

```
        pas286 block18.pas $(PASFLAGS)

block19.bl:         block19.obj rk4.obj
        submit :PFP:csd/pasbldl( block19, 'block19.obj,rk4.obj' )

block19.obj:        block19.pas
        pas286 block19.pas $(PASFLAGS)

block20.bl:         block20.obj rk4.obj
        submit :PFP:csd/pasbldl( block20, 'block20.obj,rk4.obj' )

block20.obj:        block20.pas
        pas286 block20.pas $(PASFLAGS)

block21.bl:         block21.obj rk4.obj
        submit :PFP:csd/pasbldl( block21, 'block21.obj,rk4.obj' )

block21.obj:        block21.pas
        pas286 block21.pas $(PASFLAGS)

block22.bl:         block22.obj rk4.obj
        submit :PFP:csd/pasbldl( block22, 'block22.obj,rk4.obj' )

block22.obj:        block22.pas
        pas286 block22.pas $(PASFLAGS)

block23.bl:         block23.obj rk4.obj
        submit :PFP:csd/pasbldl( block23, 'block23.obj,rk4.obj' )

block23.obj:        block23.pas
        pas286 block23.pas $(PASFLAGS)

block24.bl:         block24.obj lrk4.obj
        submit :PFP:csd/pasbldl( block24, 'block24.obj,lrk4.obj' )

block24.obj:        block24.pas
        pas286 block24.pas $(PASFLAGS)

block25.bl:         block25.obj lrk4.obj
        submit :PFP:csd/pasbldl( block25, 'block25.obj,lrk4.obj' )

block25.obj:        block25.pas
        pas286 block25.pas $(PASFLAGS)

block26.bl:         block26.obj lmrk4.obj
        submit :PFP:csd/pasbldl( block26, 'block26.obj,lmrk4.obj' )

block26.obj:        block26.pas
        pas286 block26.pas $(PASFLAGS)

block27.bl:         block27.obj lmrk4.obj
```

```
        submit :PFP:csd/pasbldl( block27, 'block27.obj,lmrk4.obj' )

block27.obj:        block27.pas
        pas286 block27.pas $(PASFLAGS)

block28.bl:         block28.obj rk4.obj
        submit :PFP:csd/pasbldl( block28, 'block28.obj,rk4.obj' )

block28.obj:        block28.pas
        pas286 block28.pas $(PASFLAGS)

block29.bl:         block29.obj rk4.obj
        submit :PFP:csd/pasbldl( block29, 'block29.obj,rk4.obj' )

block29.obj:        block29.pas
        pas286 block29.pas $(PASFLAGS)

block30.bl:         block30.obj rk4.obj
        submit :PFP:csd/pasbldl( block30, 'block30.obj,rk4.obj' )

block30.obj:        block30.pas
        pas286 block30.pas $(PASFLAGS)

block31.bl:         block31.obj rk4.obj
        submit :PFP:csd/pasbldl( block31, 'block31.obj,rk4.obj' )

block31.obj:        block31.pas
        pas286 block31.pas $(PASFLAGS)

lmrk4.obj:          lmrk4.pas
        pas286 lmrk4.pas $(PASFLAGS)

lrk4.obj: lrk4.pas
        pas286 lrk4.pas $(PASFLAGS)

rk4.obj:  rk4.pas
        pas286 rk4.pas $(PASFLAGS)

table.obj:          table.pas
        pas286 table.pas $(PASFLAGS)

crossbar.bl sequencer.bl: network.txt
        submit :PFP:csd/xbc( network.txt )

clean:
        delete *.lst,*.obj,*.mp?,*.bl

run: $(PROGRAM)
        reset
        download process.txt
        start process.txt
```

```
ioserve process.txt 30
```

```
File: NETWORK.TXT

[ Spinning Missile - Crossbar setup ]

loop
cycle [ 1 ]
    p23 := p21.2;              [ state - gp2 ]
    p27 := p25.2;              [ state - gp5 ]
    p22 := p20.2;              [ state - gt2 ]
    p26 := p24.2;              [ state - gt5 ]
    p11, p12, p13,
    p15, p16, p17,
    p19, p28, p31 := p14.2     [ state - us ]
cycle [ 2 ]
    p25 := p23.2;              [ state - gp4 ]
    p24 := p22.2;              [ state - gt4 ]
    p18, p28, p31 := p17.2;    [ state - ps ]
    p14 := p15.2;              [ state - vs ]
    p8, p9, p10 := p19.2;      [ state - z ]
cycle [ 3 ]
    p17 := p4.2;               [ table - ltf ]
    p14, p16, p18,
    p19, p20, p21,
    p22,
    p23, p28, p30 := p29.2;    [ state - theta ]
    p31 := p15.2;              [ state - vs ]
cycle [ 4 ]
    p14, p15,
    p18, p28, p30 := p31.2;    [ state - rs ]
cycle [ 5 ]
    p20, p21,
    p22, p23 := p30.2;         [ state - psi ]
    p14, p16,
    p29, p31 := p28.2;         [ state - qs ]
cycle [ 6 ]
    p14 := p1.2;               [ table - tf ]
    p15, p16, p20,
    p21, p22,
    p23, p28, p31 := p18.2;    [ state - phi ]
cycle [ 7 ]
    p15, p31 := p9.2;          [ table - wwef ]
    p14, p19, p28 := p16.2;    [ state - ws ]
cycle [ 8 ]
    p14, p15, p16,
    p17, p28, p31 := p10.2;    [ table - rhof ]
cycle [ 9 ]
    p14 := p5.2;               [ table - acd0f ]
    p15, p16,
    p28, p31 := p26.2;         [ state - fz ]
cycle [ 10 ]
    p17 := p13.2;              [ table - clpf ]
```

```
    p15, p16,
    p28, p31 := p27.2;        [ state - fy ]
cycle [ 11 ]
    p14, p16, p28 := p8.2;    [ table - wnsf ]
cycle [ 12 ]
    p15, p16 := p11.2;        [ table - acnaf ]
    p28, p31 := p2.2;         [ table - riyf ]
cycle [ 13 ]
    p17 := p12.2;             [ table - cldtf ]
    p28, p31 := p3.2;         [ table - lclcgf ]
    p14, p15, p16 := p0.2;    [ table - rmf ]
cycle [ 14 ]
    p28, p31 := p7.2;         [ table - cmqf ]
cycle [ 15 ]
    p28, p31 := p6.2;         [ table - cmaf ]
```

```
File: PROCESS.TXT
p32 block00.bl smissile.txt <null>
p33 block01.bl smissile.txt <null>
p34 block02.bl smissile.txt <null>
p35 block03.bl smissile.txt <null>
p36 block04.bl smissile.txt <null>
p37 block05.bl smissile.txt <null>
p38 block06.bl smissile.txt <null>
p39 block07.bl smissile.txt <null>
p40 block08.bl smissile.txt <null>
p41 block09.bl smissile.txt <null>
p42 block10.bl smissile.txt <null>
p43 block11.bl smissile.txt <null>
p44 block12.bl smissile.txt <null>
p45 block13.bl smissile.txt <null>
p46 block14.bl smissile.txt <null>
p47 block15.bl smissile.txt <null>
p48 block16.bl smissile.txt <null>
p49 block17.bl smissile.txt <null>
p50 block18.bl smissile.txt <null>
p51 block19.bl smissile.txt <null>
p52 block20.bl smissile.txt <null>
p53 block21.bl smissile.txt <null>
p54 block22.bl smissile.txt <null>
p55 block23.bl smissile.txt <null>
p56 block24.bl smissile.txt <null>
p57 block25.bl smissile.txt <null>
p58 block26.bl smissile.txt <null>
p59 block27.bl smissile.txt <null>
p60 block28.bl smissile.txt <null>
p61 block29.bl smissile.txt <null>
p62 block30.bl smissile.txt <null>
p63 block31.bl smissile.txt <null>
crossbar crossbar.bl <null> <null>
sequencer sequencer.bl <null> <null>
```

```
File: RK4.PAS
Module Solve_Differential_Equation;

Public Problem_Specifications;
   Procedure Initial_Conditions;
   Procedure Evaluate_Derivatives( var Y_dot : Real; Y, time : Real;
                                   first_eval : boolean );

Public Solve_Differential_Equation;

Var
    time, h, Y, Y_dot : real;
    first_eval        : boolean;

Program Solve_Differential_Equation;



Procedure Integrate;
{
          Integration Routine using Fourth Order Runge-Kutta.
                                                                    }
var
    inv_three,
    k1, k2, k3,
    half_h :     real;

begin

    inv_three := 1 / 3.0;
    half_h := 0.5 * h;
    while true do begin
        Evaluate_Derivatives(Y_dot, Y, time, true);
            k1 := Y + half_h * Y_dot;
        Evaluate_Derivatives(Y_dot, k1, time + half_h, false);
            k2 := Y + half_h * Y_dot;
        Evaluate_Derivatives(Y_dot, k2, time + half_h, false);
            k3 := Y + h * Y_dot;
        Evaluate_Derivatives(Y_dot, k3, time + h, false);
            Y := ( k1 + 2.0*k2 + k3 + half_h * Y_dot - Y ) * inv_three;
        time := time + h;
    end;

end; { PROCEDURE Integrate }


begin

    Initial_Conditions;
    Integrate;
```

```
end.
```

```
File: SMISSILE.TXT
# integration step
real_32bit
1
0.0005
```

```
File: TABLE.PAS
Module Solve_Table;

Public Problem_Specifications;
   Procedure Initialize_Table;
   Procedure Evaluate_Table( time : Real );

Public Solve_Table;

    Var    time, integration_step : Real;

Program Solve_Table;

Procedure Table_Value;

var
    half_step : real;

begin

    time := 0.0;
    half_step := 0.5 * integration_step;
    while true do begin
        Evaluate_Table( time );                      { time }
        Evaluate_Table( time + half_step );          { time+0.5*h }
        Evaluate_Table( time + half_step );          { time+0.5*h }
        Evaluate_Table( time + integration_step );   { time+h }
        time := time + integration_step;
    end;

end; { Procedure Table_Value }

begin

    Initialize_Table;
    Table_Value;

end.
```